MASTER EN SCIENCES INFORMATIQUES

Etude de la traduction en SQL des hiérarchies de spécialisation

Denil, Caroline

Award date:
1998

FACULTÉS UNIVERSITAIRES NOTRE-DAME DE LA PAIX,

NAMUR

INSTITUT D'INFORMATIQUE

RUE GRANDGAGNAGE, 21, B-5000 NAMUR (BELGIUM)

Étude de la traduction en SQL des hiérarchies de spécialisation

Caroline DENIL

Mémoire présenté en vue de l'obtention du grade de

Licencié en Informatique

Année Académique 1997 - 1998

Résumé

Les relations is-a sont des constructions de base proposées dans la plupart des modèles conceptuels de systèmes d'information. Mais, très peu de systèmes de gestion de bases de données permettent de les représenter complètement. Ce mémoire propose d'analyser les relations is-a et de les transformer en constructions conformes au modèle relationnel. Nous utilisons trois techniques : la matérialisation, l'héritage ascendant et l'héritage descendant. Les schémas obtenus après transformation sont en SQL. Le but de ce mémoire est de permettre la génération de ces codes SQL dans l'atelier DB-Main.

Abstract

Is-a relations are basic constructs proposed in most information system conceptual models. On the other side, few database management systems can give an explicit and complete representation. The aim of this thesis is to analyze is-a relations in details, and to propose correct techniques to express is-a relations into relational constructs. The resulting schemas are expressed into SQL. We propose the generation of SQL code into the DB-Main Case Tool.

Mes plus vifs remerciements vont à Monsieur le professeur Jean-Luc Hainaut, promoteur de ce mémoire, qui a eu le courage et la patience de m'aider depuis la spécification jusqu'à la réalisation de ce mémoire.

Merci également à Jean-Marc et à l'équipe DB-Main pour l'aide et les conseils qui m'ont été fortement utiles, ainsi que pour leur disponibilité.

Je voudrais aussi remercier toutes les personnes présentes à mes côtés lors de la rédaction, parfois pénible, de ce mémoire : Christophe pour la relecture et ses bons conseils en grammaire ainsi que pour son soutien quotidien, et ma soeur pour la correction d'un texte qui lui semblait si étrange. Merci encore à mes parents de m'avoir soutenue pendant toutes mes années d'étude.

Merci aussi à toutes les personnes dont je ne pourrais citer le nom et qui ont contribué à mettre une ambiance dans le pool et dans les auditoires. Sans eux, la vie quotidienne aurait été moins agréable.

Et enfin, je ne pourrais oublier Virginie qui m'a beaucoup aidée et soutenue pendant ces deux années, deux années qui resteront longtemps dans ma mémoire.

I. Introduction

II. Analyse des relations is-a

III. Transformations possibles dans DB-Main

IV. Modification du générateur SQL

I.INTRODUCTION

1. Contexte

1.1 Cycle de vie d'une base de données[8],[2]

Le cycle de vie d'une base de données, (Figure 1), commence par l'étude des besoins ou l'étude d'opportunité. Celle-ci prépare un avant-projet de solution à partir des besoins exprimés par l'organisation. Il s'agit de déterminer le domaine dans lequel l'organisation va travailler.

Ensuite, vient l'analyse conceptuelle, qui permet de passer de l'informel au formel. Sur base de l'avant-projet, elle élabore une solution fonctionnelle détaillée mais indépendante de tout moyen de réalisation. Il sort de cette analyse un schéma conceptuel. Le modèle choisi pour représenter le schéma conceptuel est le modèle entité-association. Une brève présentation en est donnée au paragraphe 2.

La conception logique produit un schéma, le schéma logique, sémantiquement équivalent au schéma conceptuel (Figure 2). Ce schéma doit être conforme au modèle du Système de Gestion de Données choisi pour réaliser la base de données.

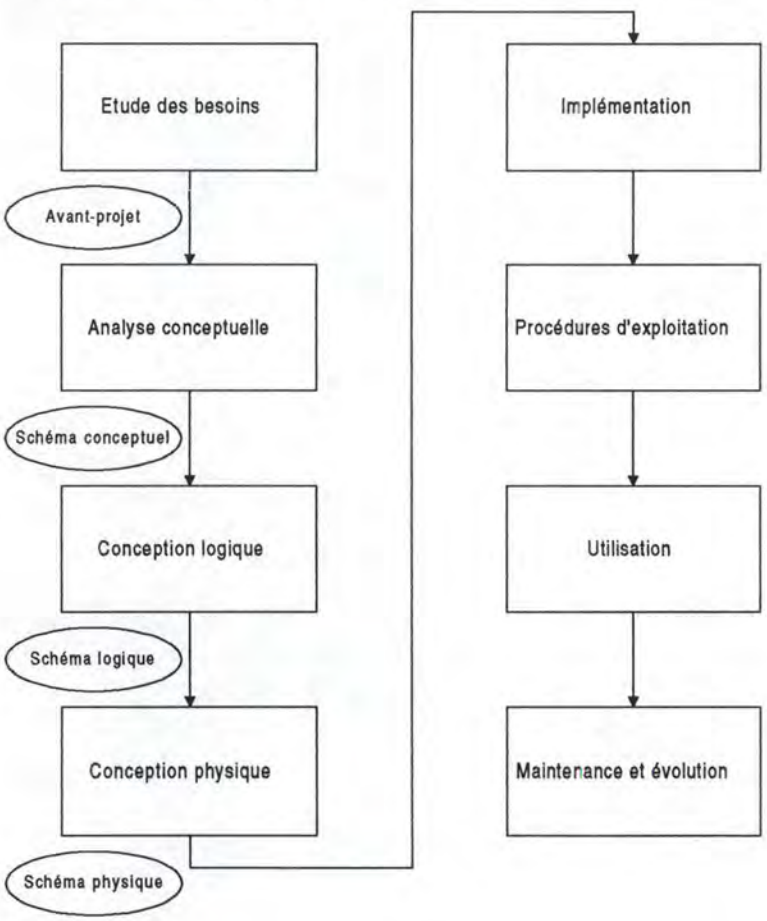


Figure 1 : Cycle de vie d'une base de données

Le modèle du SGD qui nous sert à représenter la base de données est le modèle relationnel.

La conception physique transforme la solution logique, y ajoute des paramètres pour rendre le schéma opérationnel et efficace.

Dans notre cas, le schéma physique nous permet de générer une traduction SQL.

La phase d'implémentation est la phase de chargement des données.

Les dernières phases du cycle de vie concernent la gestion globale de la base de données, son utilisation ainsi que sa maintenance et son évolution.

1.2 Conception logique[2],[3]

Le niveau logique est concerné par la modélisation de structures logiques de données.

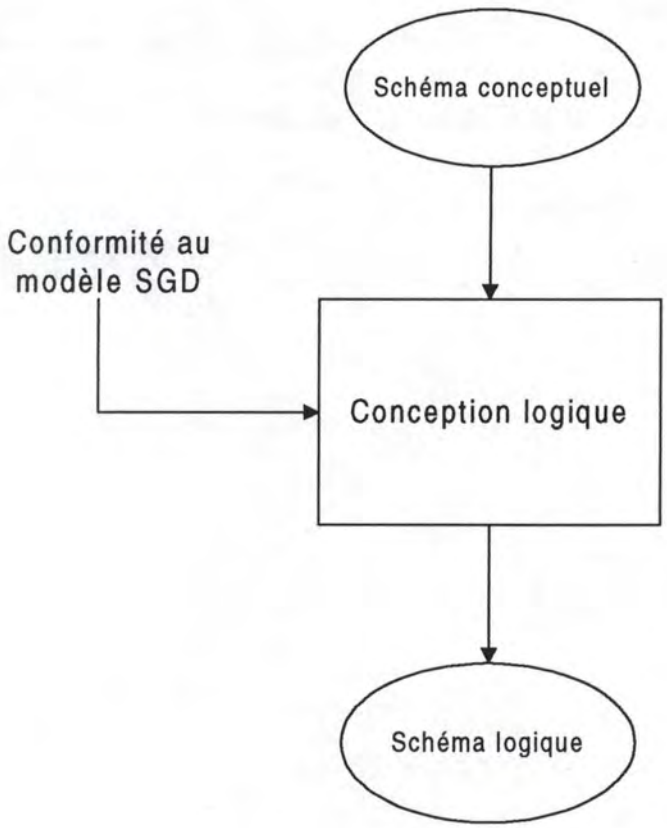


Figure 2 : Conception logique

Certaines structures ne sont pas admises dans le modèle relationnel comme, par exemple, les types d'associations. La liste des structures non conformes au modèle relationnel est présentée au Tableau 1. Ces structures seront expliquées dans le paragraphe 2.

Structures de généralisation/spécialisation
Types d'associations
Type d'entités sans attributs
Attributs décomposables
Attributs multivalués
Contraintes autres que les identifiants et les contraintes référentielles

Tableau 1 : Structures non conformes au modèle relationnel

Pour passer d'un schéma conceptuel à un schéma logique, des transformations de schéma sont utilisées. Ces transformations doivent être réversibles, c'est-à-dire qu'elles produisent des schémas décrivant le même domaine d'application que les schémas initiaux. Si la suite des transformations jouit de la propriété d'équivalence (le schéma produit est équivalent au schéma source) et de complétude (toute la sémantique du schéma source a été traduite), deux sortes de spécifications sont produites :

- des structures conformes au modèle logique;
- des contraintes additionnelles conformes ou non au modèle logique.

Diverses techniques permettent l'implémentation de ces composants que l'on ne peut pas traduire.

Les checks en SQL permettent d'exprimer de tels composants : chaque contrainte est exprimée sous la forme d'un prédicat associé à une colonne, à une table ou au schéma. Le prédicat est évalué après toute opération d'insertion et de modification de lignes d'une table. Si le prédicat n'est pas vérifié, l'opération est annulée.

Dans certains systèmes de gestion de bases de données, comme ORACLE, le prédicat check ne peut faire intervenir que le contenu de la ligne courante.

D'autres méthodes permettent le codage d'un schéma. C'est le cas des procédures déclenchées ou triggers. Une procédure définit la réaction à adopter en cas de mise à jour. En cas de modification, si la précondition est vérifiée, la procédure est déclenchée. Cette méthode est plus puissante mais plus complexe que celle des checks.[8]

1.3 Situation du problème [1],[2],[10]

Dans les structures incompatibles avec le modèle relationnel, nous trouvons les structures de généralisation/spécialisation ou relations is-a. La généralisation est un mécanisme d'abstraction en fonction duquel une relation entre des classes d'objets est considérée comme une classe d'objets génériques de plus haut niveau. Par exemple, le type d'entités Personnel peut être considéré comme un type générique (un surtype) par rapport aux types d'entités Ouvrier, Cadre, Employé considérés comme types spécifiques (ou sous-types). Chaque entité du sous-type est une entité du surtype [2].

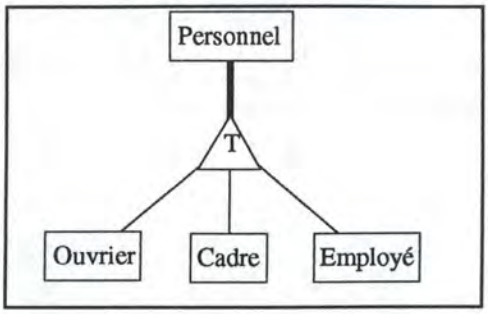


Figure 3 : Exemple de relation is-a

Dans une relation is-a, les sous-types héritent de tous les rôles et des contraintes d'intégrité de leur surtype.

La plupart des Systèmes de Gestion de Base de Données ne donnent pas de constructions logiques correspondant à la relation is-a. En fait, la plupart des auteurs proposent des transformations incorrectes ou incomplètes, ignorant les contraintes d'intégrités importantes.

Les trois techniques de base proposées pour la transformation de la relation is-a sont :

- Matérialisation is-a : chaque sous-type est représenté par une entité indépendante reliée au surtype par une association one-to-one (Figure 4.1) ;
- Héritage ascendant : seul le surtype est représenté (Figure 4.2) ;
- Héritage descendant : seuls les sous-types sont représentés (Figure 4.3). Le type d'entités A s'il est présent, réunit les entités n'appartenant pas aux sous-types B et C.

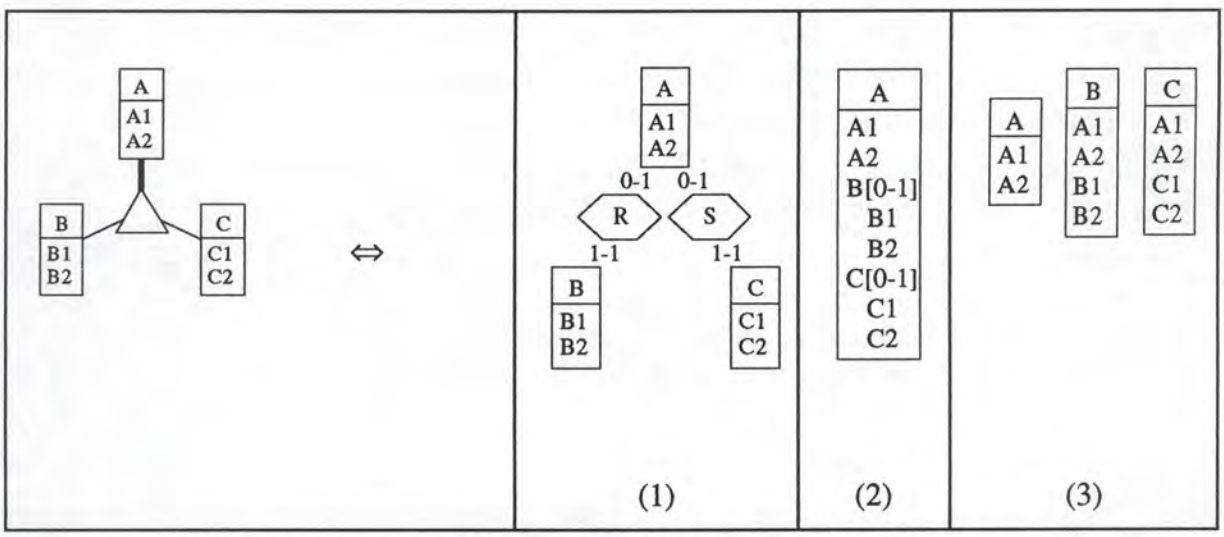


Figure 4: Trois techniques standards pour remplacer la relation is-a

Dans les techniques (1) et (2), certains auteurs incluent un attribut TYPE qui indique à quel sous-type chaque entité appartient.

Les surtypes de relations is-a peuvent être soumis à des contraintes de disjonction, de totalité et de partition mais la plupart des auteurs ignorent ces contraintes qui compliquent les schémas. De plus, les rôles joués par les sous-types et surtype ainsi que d'autres contraintes, comme les identifiants, amènent à des schémas plus complexes encore.

Il semble dès lors nécessaire d'automatiser la transformation de relations is-a et d'en donner une représentation logique complète et correcte.

1.4 *But du mémoire*

Par cette étude, nous voulons proposer un moyen de transformer automatiquement les relations is-a en des structures compatibles avec le modèle relationnel. Pour cela, nous allons étudier différentes relations is-a, leur intérêt et leur élimination par les trois techniques de base pour obtenir un schéma relationnel, puis le codage en SQL qui en découle. Ces relations is-a sont soumises à des contraintes de disjonction, de totalité et de partition.[1]

Nous présenterons ensuite quelques cas de figures, les solutions de transformation possibles et leurs justifications, ainsi que les modifications que l'on peut apporter aux autres cas de figures pour permettre leur transformation en relationnel.

L'atelier DB-Main devra être modifié pour permettre la génération des transformations des relations is-a et la génération du code SQL qui en découle. Ce code est prévu pour être implémenté sur Interbase.

1.5 *Démarche générale*

Ce mémoire débute par une analyse détaillée des relations is-a. Cette étude nous permet de ne conserver que des cas généraux et de tirer les éléments qu'il faut ajouter aux autres schémas pour qu'ils puissent être transformés.

Grâce aux cas généraux, nous pouvons déterminer le code SQL, complété de checks ou de triggers.

Munis de tous ces éléments, il nous est dès lors possible de programmer les transformations dans l'atelier DB-Main pour qu'il puisse, de manière automatique, donner des schémas équivalents aux schémas initiaux et en donner le code SQL.

[1] Ces termes seront définis plus loin.

2. *Modèle entité-association*[2],[8],[11]

Le modèle entité-association est un modèle qui permet d'exprimer la sémantique des données à l'aide des concepts d'entité, d'association, d'attribut et du mécanisme des contraintes d'intégrité.

2.1 *Concepts*[2]

Type d'entités

Une entité est une chose concrète ou abstraite appartenant au réel perçu à propos de laquelle on veut enregistrer des informations. Une entité peut posséder des attributs.

Un type d'entités est une classe de toutes les entités possibles du réel perçu qui vérifient la définition constitutive du type. Les types d'entités ne forment pas nécessairement des classes disjointes.

Type d'associations

Une association est définie par une correspondance entre deux ou plusieurs entités, non nécessairement distinctes, où chacune assume un rôle donné.

Un type d'associations est la classe de toutes les associations possibles du réel perçu qui vérifient la définition constitutive du type.

Attribut

Un attribut est la caractéristique ou qualité d'une entité ou d'une association. Il peut prendre une ou plusieurs valeurs ou groupe de valeurs.

Un attribut peut être monovalué ou multivalué. Il est monovalué si pour une entité ou une association, il ne peut prendre qu'une seule valeur. Par contre, il est multivalué s'il peut prendre plusieurs valeurs d'un même type.

Un attribut peut également être atomique ou décomposable. Il est décomposable si à une entité ou une association, il fait correspondre un groupe de valeurs de types différents et peut être décomposé, au plus, en autant d'attributs qu'il y a de types différents dans le groupe de valeurs.

Un attribut peut également être obligatoire ou facultatif. Il est obligatoire s'il doit prendre une valeur effective pour chaque occurrence du type qu'il caractérise.

Contrainte d'intégrité

Le modèle Entité-Association a une capacité limitée de représentation du réel : il ne permet pas de spécifier toutes les propriétés sémantiques. Ces propriétés, appelées contraintes d'intégrité ne sont pas représentées par les concepts de base du modèle à savoir les types d'entités, les types d'associations et les attributs.

Une contrainte particulière, représentée dans le modèle entité-association est la contrainte de connectivité ou de cardinalité. La connectivité d'un type d'associations est définie par un ensemble de couples d'entiers (min_i, max_i),

- où min_I indique le nombre minimum de fois que, à tout moment, toute occurrence de l'entité E_i doit assumer le rôle ro_I, c'est-à-dire le nombre minimum d'occurrences du type d'associations auquel toute occurrence du type d'entités E_i doit participer.
- où max_i indique le nombre maximum de fois que, à tout moment, toute occurrence de l'entité E_i doit assumer le rôle ro_I, c'est-à-dire le nombre maximum d'occurrences du type d'associations auquel toute occurrence du type d'entités E_i doit participer.

Une autre contrainte d'intégrité essentielle est l'identifiant d'un type d'entités ou d'un type d'associations. Il permet de repérer de manière univoque chaque occurrence de ce type.

Groupe

Un groupe représente une construction attachée à un objet parent par exemple à un type d'entités, à un type d'associations ou à un attribut. Il est employé pour représenter des concepts comme les identifiants, les clés étrangères, les index, les groupes d'attributs coexistants ou exclusifs... Un groupe est constitué d'attributs, de rôles et/ou d'autres groupes.

Relation is-a

La relation is-a ou généralisation est un mécanisme d'abstraction en fonction duquel une relation entre des classes d'objets est considérée comme une classe d'objets génériques de plus haut niveau. Les objets spécifiques (sous-types) héritent des propriétés de l'objet générique (surtype). Si E_j est un sous-type de E_1 alors toute occurrence e_{js} est aussi une occurrence de E_1 dont elle hérite les propriétés, c'est-à-dire les attributs et les associations auxquelles cette occurrence de E_1 participe. Deux propriétés peuvent venir compléter les types d'entités compris dans une relation is-a : la disjonction et la totalité. La disjonction entre sous-types indique que chaque entité du surtype appartient à au plus un des sous-types. Dans le cas contraire, il y a recouvrement entre les sous-types. La contrainte de totalité indique que chaque entité du surtype doit appartenir à au moins un sous-type.

Quand le type d'entités surtype est soumis à une contrainte de disjonction et de totalité, la contrainte peut être résumée en une contrainte de partition.

2.2 *Représentation*[2]

Dans les schémas repris dans notre étude, un type d'entités est représenté par un rectangle comportant un cartouche où figure le nom du type d'entités.

Un type d'associations est représenté par un hexagone relié par des segments de droites aux types d'entités sur lesquels est défini le type d'associations. Dans le cartouche supérieur de l'hexagone, on indique le nom du type d'associations et sur la patte qui relie un type d'entités à l'hexagone, on indique le nom du rôle joué par le type d'entités dans le type d'associations.

Le nom d'un attribut d'un type d'entités ou d'un type d'associations est inscrit dans la partie inférieure du rectangle ou de l'hexagone qui représente le type d'entités ou le type d'associations (Figure 5).

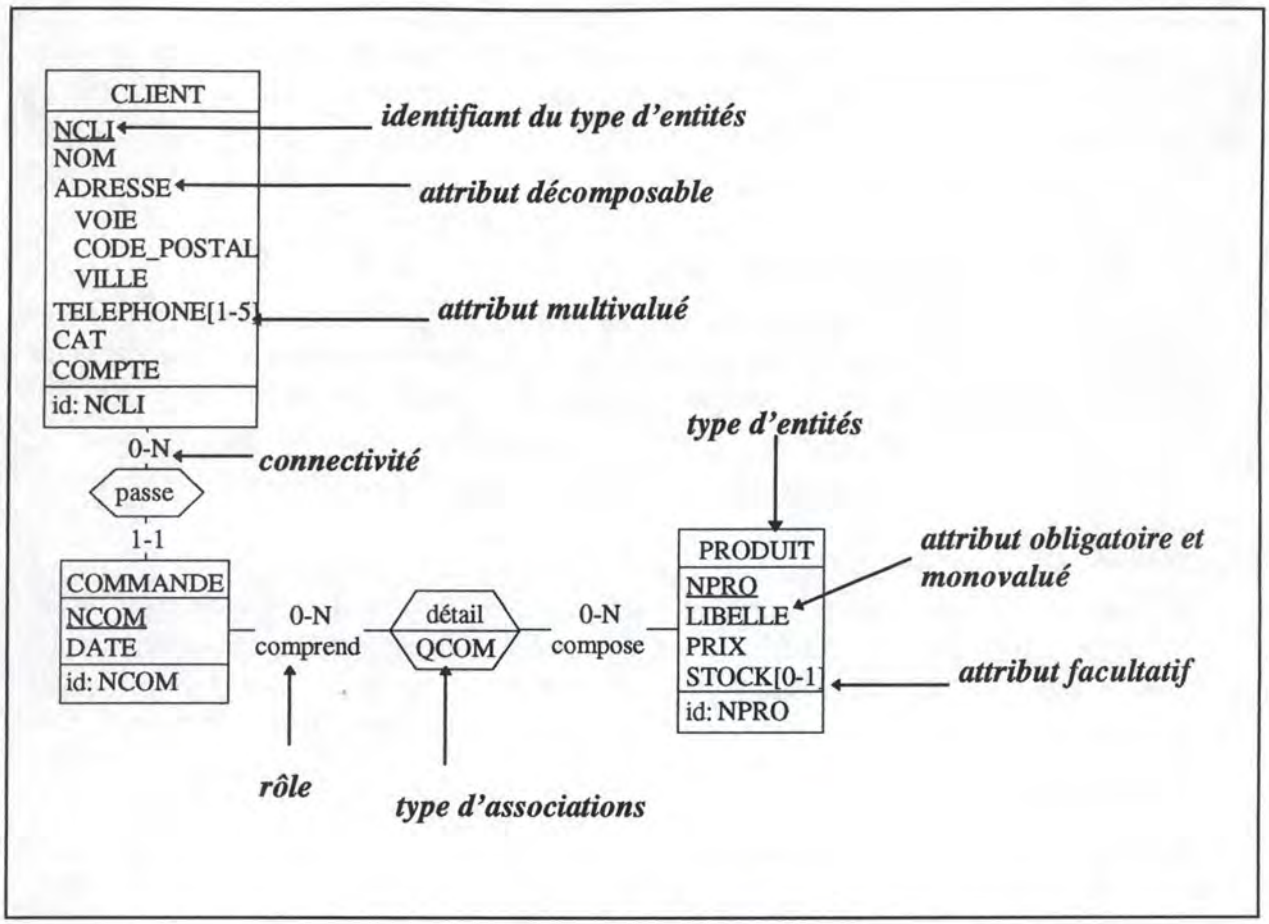


Figure 5 : Représentation du schéma Entité-Association

La relation is-a est représentée par un triangle avec une patte plus grasse reliant la relation au surtype, les autres pattes étant dirigées vers les sous-types (Figure 6).

Dans le triangle, il est commun de noter la contrainte du surtype, à savoir :

- D pour la contrainte de disjonction entre sous-types
- T pour la contrainte de totalité entre sous-types
- P pour la contrainte de partition entre sous-types.

La relation is-a de la figure 6 1 exprime le fait qu'un élève peut mais ne doit pas être inscrit en mathématique ou en sciences.

La contrainte de disjonction exprime le fait qu'un surtype ne peut appartenir qu'à un seul sous-type. Par exemple (Figure 6 2), les deux classes qui nous intéressent sont oiseau et poisson. Ces classes sont disjointes. Un animal est un poisson ou bien un oiseau mais il peut aussi être autre chose, comme un reptile...

La contrainte de totalité implique qu'un surtype doit obligatoirement appartenir à un sous-type au moins. Par exemple (Figure 6 3), un travailleur doit être salarié ou indépendant, mais il peut être salarié et indépendant.

La contrainte de partition implique qu'un surtype doit obligatoirement appartenir à un et un seul sous-type. Un être humain est obligatoirement soit un homme, soit une femme (Figure 6 4).

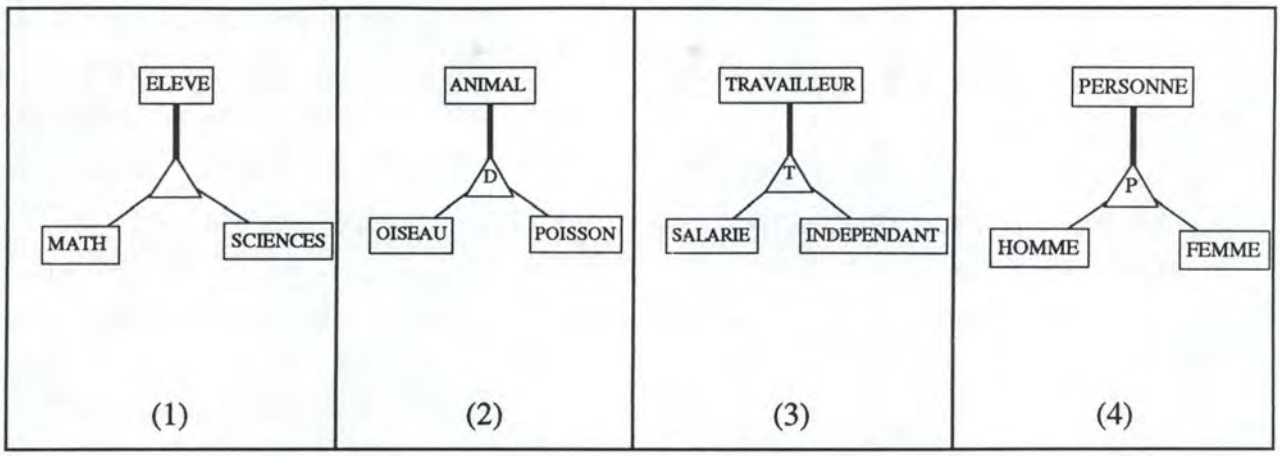


Figure 6 : Représentation de la relation is-a

2.3 *Transformations de schémas*

Une transformation de schéma est une opération qui consiste à remplacer dans un schéma S une construction C par une construction C' amenant à un nouveau schéma S'. La Figure 7 montre un exemple de transformation d'un type d'entités (T.E.) en type d'associations (T.A.).

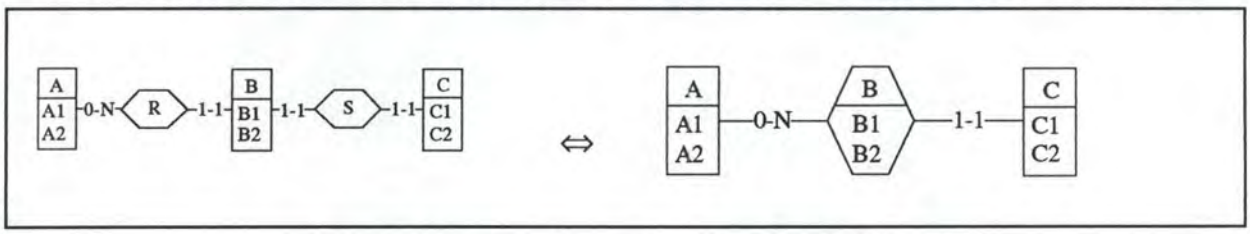


Figure 7 : Transformation d'un T.E. en T.A.

Certaines transformations augmentent la sémantique d'un schéma (ajouter un type d'entités), d'autres la diminuent (supprimer un attribut). Il existe aussi des transformations qui préservent la sémantique d'un schéma , c'est-à-dire qui sont telles que S et S' décrivent le même univers du discours (remplacer un T.E. par un T.A.). Ces dernières transformations sont appelées réversibles. Le Tableau 2 reprend des transformations pour la production d'un schéma conforme à SQL.

Structures non conformes à SQL	Proposition de transformation
Structures de généralisation/spécialisation	Transformer en types d'associations one-to-one (+ contraintes éventuelles)
Types d'associations non fonctionnels	Transformer en types d'entités
Attributs décomposables monovalués	Désagréger
Attributs multivalués	Transformer en type d'entités par représentation des instances
Types d'associations fonctionnels	Transformer en attribut de référence (+ contraintes d'inclusion éventuelles)
Contraintes autres que les identifiants et les contraintes référentielles	Les exprimer sous forme de contraintes additionnelles
Type d'entités sans attribut	Ajouter un identifiant technique

Tableau 2: Transformations pour la production d'un schéma conforme à SQL

3. *Atelier DB-Main*[2],[5],[6],[11],[12]

DB-Main est un outil Case générique consacré à l'ingénierie des applications de base de données et en particulier à la conception, au reverse engineering, à la ré-ingénierie, à l'intégration, à la maintenance et à l'évolution des bases de données.

L'outil DB-Main offre les fonctions de base pour introduire et pour gérer des bases de données, pour permettre la présentation de ces schémas selon différents formats, pour produire des rapports, pour transformer et analyser des schémas, pour produire des schémas relationnels, COBOL, CODASYL et générer des programmes SQL, COBOL...

La version courante, version 3.04, offre plusieurs fonctions originales et puissantes qui sont encore absentes dans les outils CASE traditionnels. Par exemple une boîte d'outils de transformation bien complète, des assistants de restructuration et d'analyse des schémas, un repository puissant basé sur un langage 4GL (*Voyager 2*) pour développer des rapports définis par l'utilisateur et pour générer du code. L'atelier permet encore la rétro-ingénierie, de fichiers SQL par exemple, et l'analyse de textes sources de programmes. Une dernière fonction de base est l'enregistrement et le replay d'activités de design.

3.1 *Architecture de l'atelier*

L'architecture de DB-Main est basée sur les composants suivants :

- une interface graphique simple à utiliser;
- des assistants qui sont des outils d'aide à la résolution de problèmes propres et spécifiques (de transformations, de conformité, de rétro-ingénierie...); pour résoudre des problèmes complexes et répétitifs, l'analyste peut utiliser des scripts prédéfinis ou personnalisés;
- la machine Voyager, interpréteur du langage *Voyager 2*, qui exécute des fonctions personnalisées développées par l'analyste;
- les outils de base qui permettent l'accès au référentiel, la gestion de son contenu, l'importation/exportation de spécification, la transformation de schéma, l'analyse de texte...

3.2 *Approche transformationnelle*

Dans le domaine du génie logiciel, la conception d'un composant est souvent modélisée comme une séquence de transformations. Dans le domaine des bases de données, la conception de schéma peut être définie comme une suite de modifications de structures de données. On parle alors d'approche transformationnelle car toutes les modifications appliquées au schéma sont considérées comme des transformations et le processus de conception de bases de données est modélisé comme une séquence de transformations de schéma. DB-Main s'appuie sur une telle approche : l'atelier propose des boîtes à outils de transformations, des assistants à la transformation qui permettent d'automatiser les transformations que l'on peut appliquer au schéma courant.

3.3 *Générateur SQL de DB-Main*

Le générateur SQL est un programme écrit en *Voyager 2* qui permet la génération du code SQL correspondant au schéma courant dessiné dans DB-MAIN.

Ce code SQL est conforme à un certain système de gestion de bases de données. Le SGBD choisi dans ce mémoire propose un code SQL, où les clés primaires, secondaires et les checks sont exprimés. Le code SQL des checks peut contenir des références à d'autres tables.

Nous nous proposons de modifier ce générateur SQL pour qu'il génère en plus certaines contraintes.

II. ANALYSE DES RELATIONS IS-A

Introduction

La première partie de ce chapitre présente différentes transformations de composants de schémas. La deuxième partie est une analyse préliminaire qui nous permet de catégoriser les relations is-a étudiées. Ensuite, dans la troisième partie, nous effectuons une analyse plus poussée de relations is-a. Nous allons chercher des transformations réversibles qui pourraient être appliquées sur ces relations is-a, voir les contraintes qui découlent des transformations et en déduire le code SQL.

Nous nous efforcerons de ne pas nous attarder sur des transformations déjà développées ou inutiles.

Un résumé des différentes transformations, des contraintes et du code SQL à générer est présenté en fin de chapitre. Nous conclurons ensuite cette étude par une illustration des transformations étudiées : deux exemples permettront une compréhension plus aisée de ces transformations.

Grâce à cette étude, nous pourrons par la suite programmer dans l'atelier DB-Main les transformations is-a et programmer la génération du code SQL correspondant.

1. Transformations de composants de schéma non relationnels en composants relationnels.[8],[9],[10]

1.1 Transformation de type d'associations (T.A.) one-to-one

Trois techniques sont proposées pour transformer les T.A. one-to-one :

- Ajouter à un type d'entités B une clé étrangère obligatoire, identifiante et référençant un type d'entités A, si celle-ci possède un identifiant (Figure 1 1) ;
- Si le type d'entités B possède un identifiant, le type d'associations. R peut être traduit en une clé étrangère inverse. Cette clé étrangère est facultative et identifiante et une contrainte d'inclusion inverse est créée (equ) (Figure 1 2) ;
- Fusionner le type d'entité B avec le type d'entité A.(Figure 1 3) ;

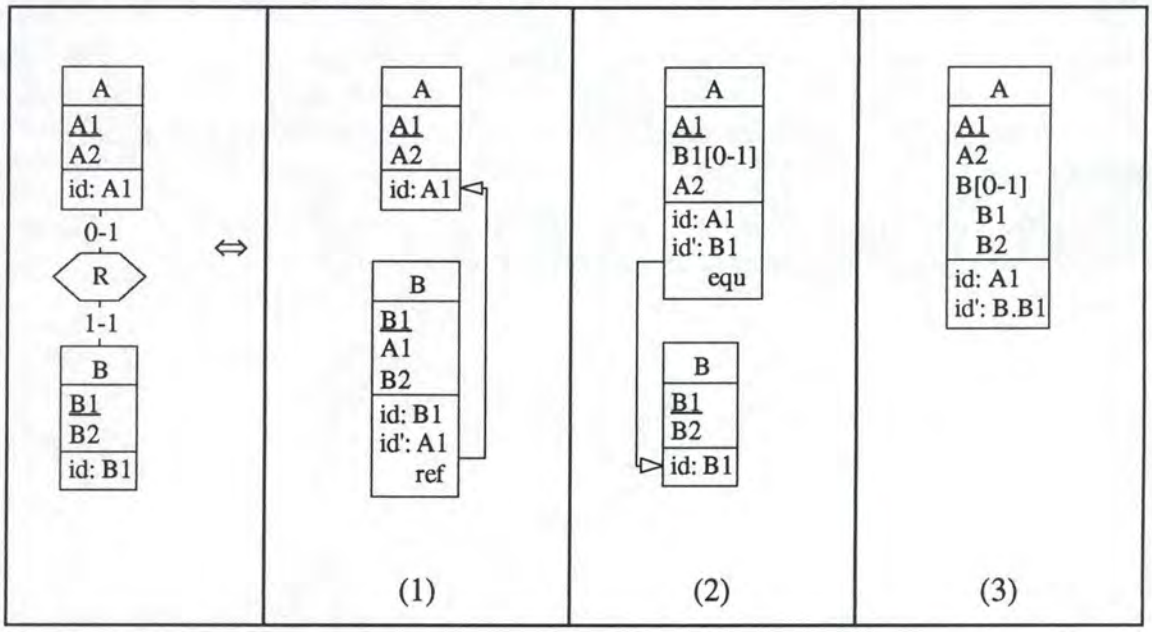


Figure 1 :Techniques de transformation de T.A. one-to-one

1.2 Transformation d'attributs décomposables et facultatifs

Trois techniques sont proposées pour la transformation d'attributs décomposables et facultatifs :

- Extraire l'attribut décomposable facultatif pour former un type d'entités (T.E.) et un type d'associations (T.A.) one-to-one. Cette technique ne sera pas utilisée parce qu'elle produit une construction non relationnelle (Figure 2 1) ;
- Désagréger l'attribut, le remplacer par ses composants et ajouter une contrainte de coexistence entre ses composants (Figure 2 2) ;
- Représenter l'attribut B par un attribut B' atomique, dont la valeur est construite par la concaténation des valeurs des composants de B (Figure 2 3).

Cette transformation n'est pas applicable si les composants individuels de B sont inclus dans des contraintes d'intégrité.

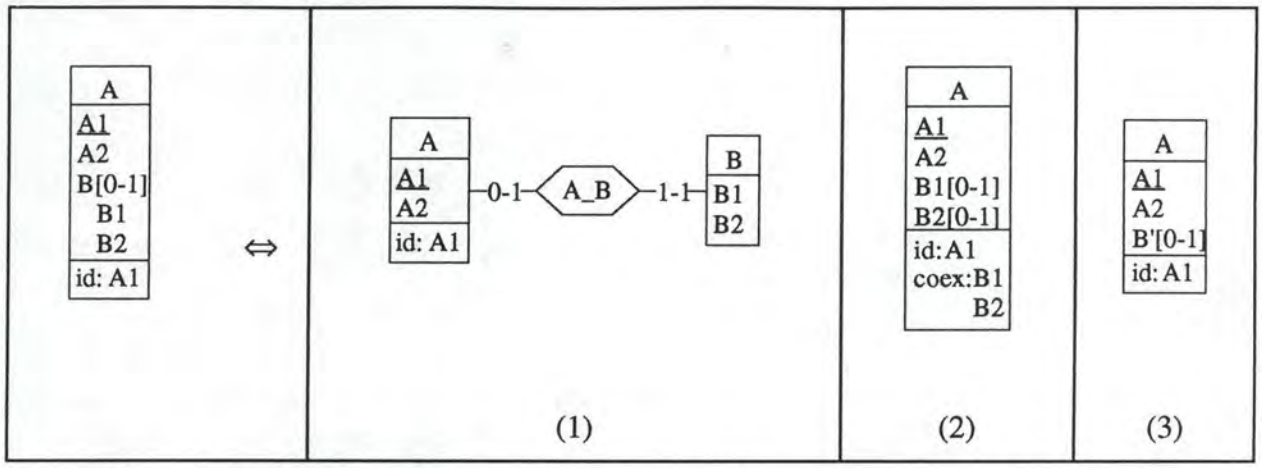


Figure 2 : Techniques de transformation d'attributs décomposables facultatifs

1.3 Transformation d'attributs multivalués

Quatre techniques sont proposées pour la transformation d'attributs multivalués :

- Extraire l'attribut pour former un T.E. et un T.A. one-to-one. On représente les instances (Figure 3 1).
- Extraire l'attribut pour former un T.E. et un T.A. one-to-many. On représente les valeurs (Figure 3 2).
- Représenter l'attribut A2 par un attribut A2' monovalué, dont la valeur est construite par la concaténation des valeurs de A2 (Figure 3 3).

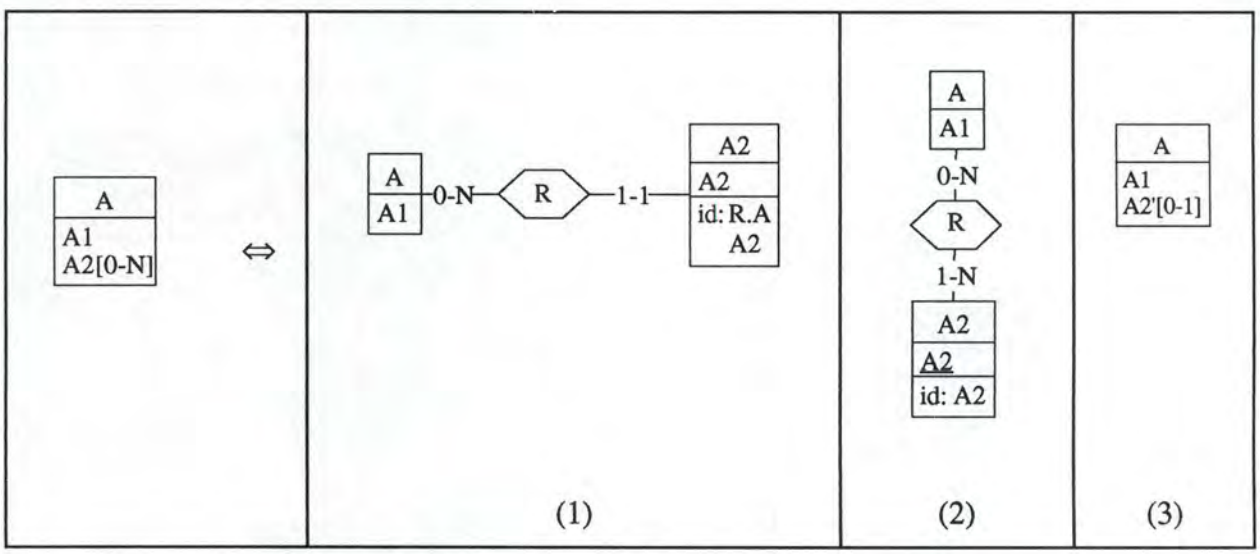


Figure 3 : Transformation d'un attribut multivalué

1.4 *Transformation de contraintes exclusive, at-least-1 et exactly-1 sur des types d'associations*

Nous devons distinguer deux situations : celle où les T.A. sont traduits en clés étrangères dans les sous-types, et celle où les T.A. sont traduits en clés étrangères dans le surtype.

Dans le premier cas, les contraintes sur les T.A. sont traduites en contraintes sur les identifiants des sous-types (Figure 4). Les interprétations en sont :

exclusive=>`disjoint(sstype1.id_sstype1,sstype2.id_sstype2…)`

at-least-1=>`surtype.id_surtype in (sstype1.id_sstype1 ∪ sstype2.id_sstype2…)`

exactly-1=>*exclusive* et *at-least-1*

Les contraintes reprises dans les figures sont exprimées dans un code proche de la syntaxe de la logique des prédicats. Mais il faut en plus considérer la fonction pop(X) comme une fonction qui donne toutes les instances d'un type X et b.ID comme étant l'attribut ID d'une instance d'un type d'entités b.

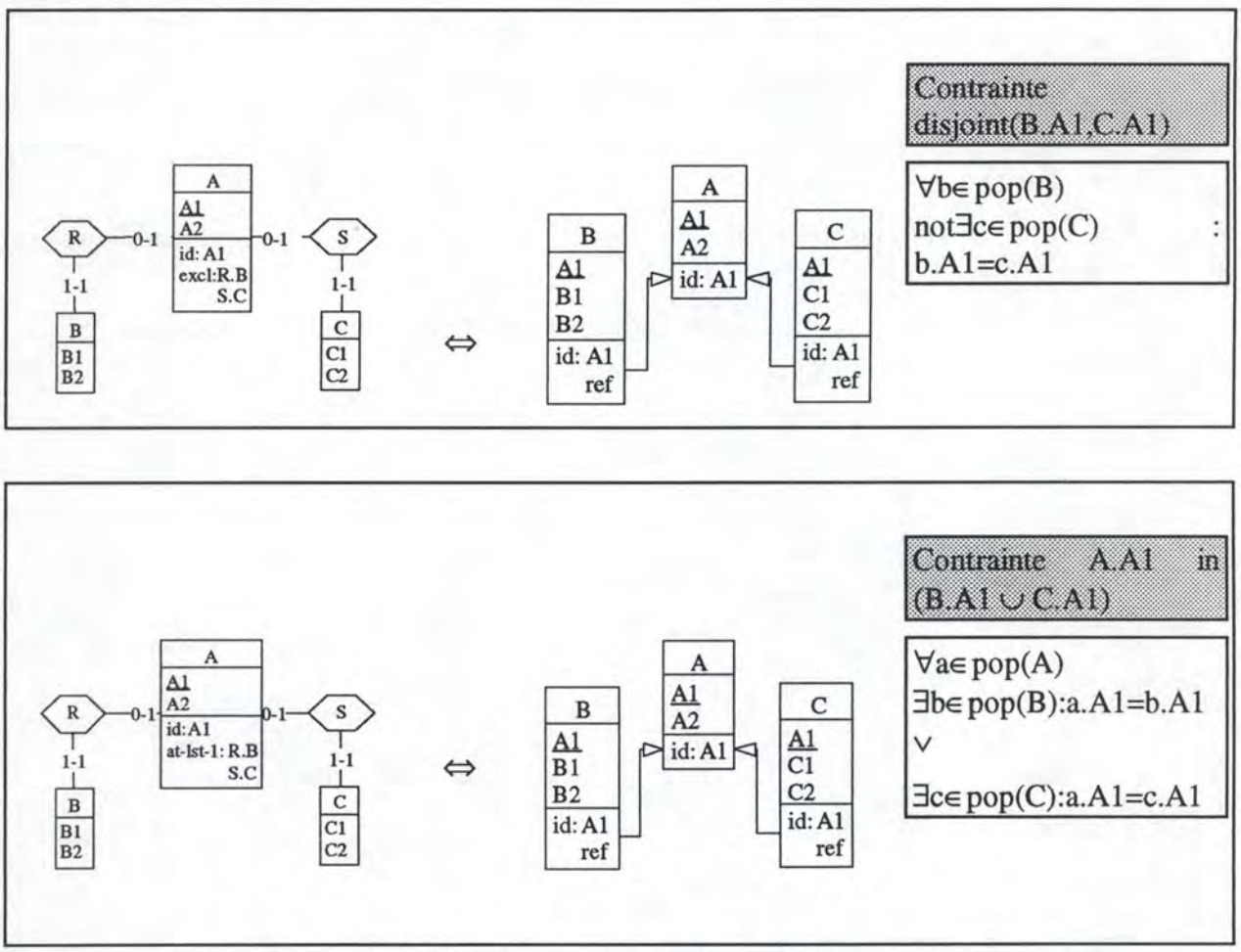


Figure 4: Transformation de contraintes sur les types d'associations (1)

Cette méthode présente un inconvénient : pour pouvoir exprimer la contrainte, il faut inclure dans la génération du code SQL un check. Il reprend des clauses qui mentionnent d'autres tables. Ces structures ne sont pas toujours acceptées par les SGBD.

Dans le cas suivant, il est possible d'éviter les contraintes inter-entité à condition de définir d'autres contraintes. Elles doivent garantir que la valeur de la clé étrangère est la copie de l'identifiant de A (Figure 5). Cette transformation présente l'intérêt d'amener des checks acceptés par la plupart des SGBD. Mais à cause de la contrainte de type equ, il est nécessaire d'utiliser une transaction pour introduire les données. Une transaction permet de faire passer la base de données d'un état cohérent à un autre état cohérent sans préserver la cohérence aux points intermédiaires.

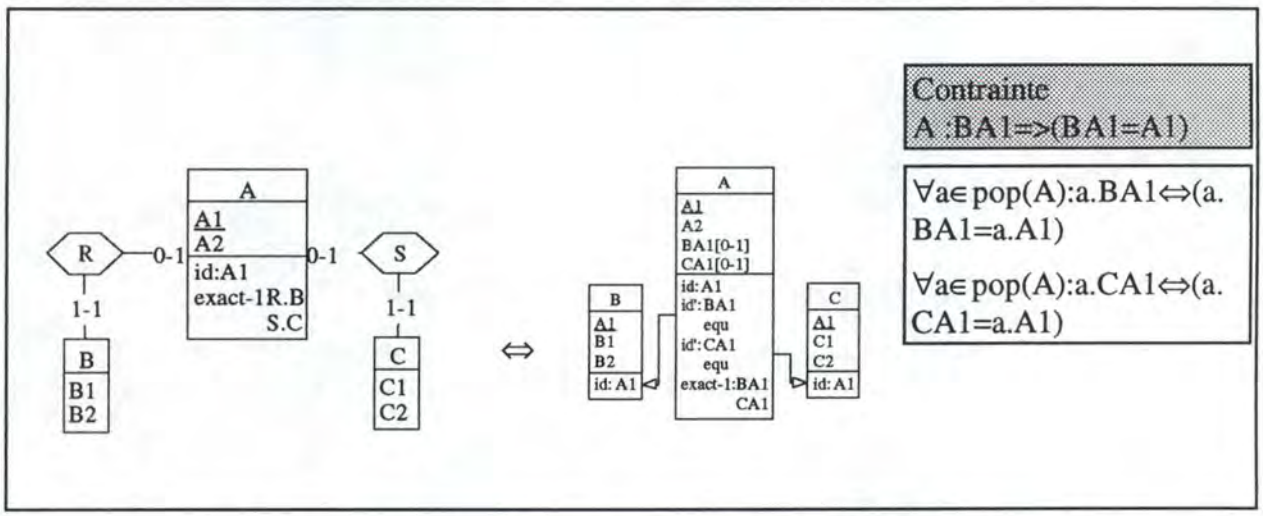


Figure 5: Transformation de contraintes sur les types d'associations (2)

Une autre méthode de transformation de T.A. consiste en la création de clés étrangères référençant les identifiants secondaires (Figure 6). Un trigger est nécessaire pour permettre l'introduction de données dans les T.E. B et C. Ce trigger met dans l'attribut BA1 (respectivement CA1), l'identifiant de A lors de l'introduction de données dans B (respectivement C).

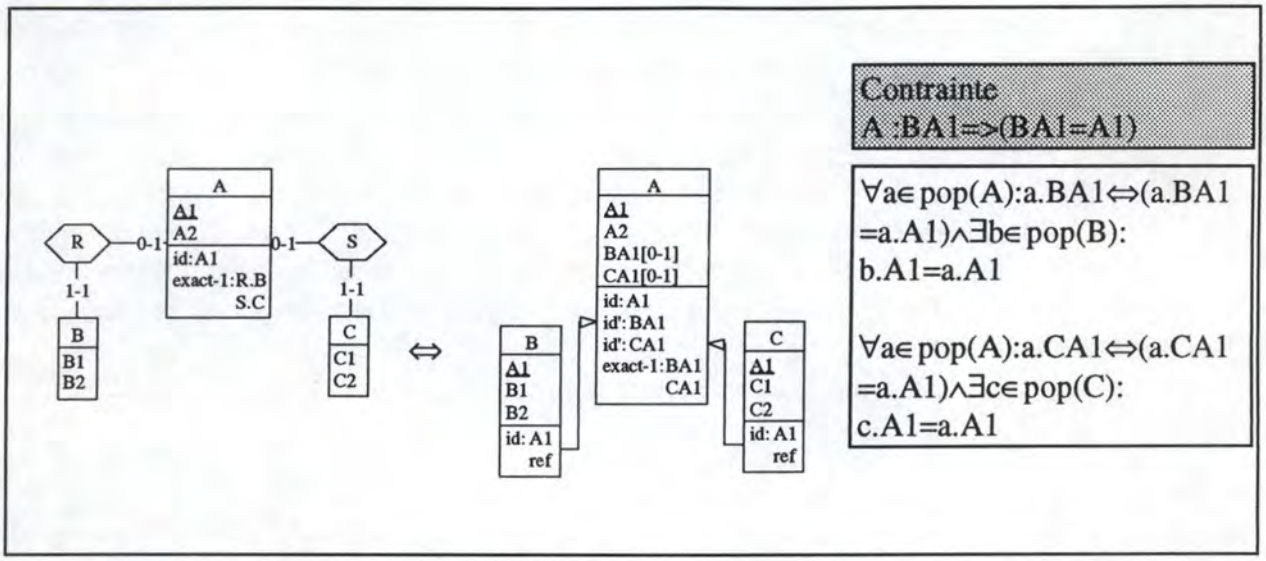


Figure 6: Transformation de contraintes sur les types d'associations (3)

1.5 *Transformation de type d'associations one-to-many*

La Figure 7 montre la transformation d'un T.A. one-to-many en clés étrangères. Cette transformation nécessite la présence d'un identifiant dans le T.E. de destination de la clé étrangère. Elle crée un attribut dans le T.E. source de la clé étrangère, référençant le T.E. de destination de la clé étrangère. La cardinalité de cet attribut est la cardinalité du rôle joué par le T.E. source de la clé étrangère.

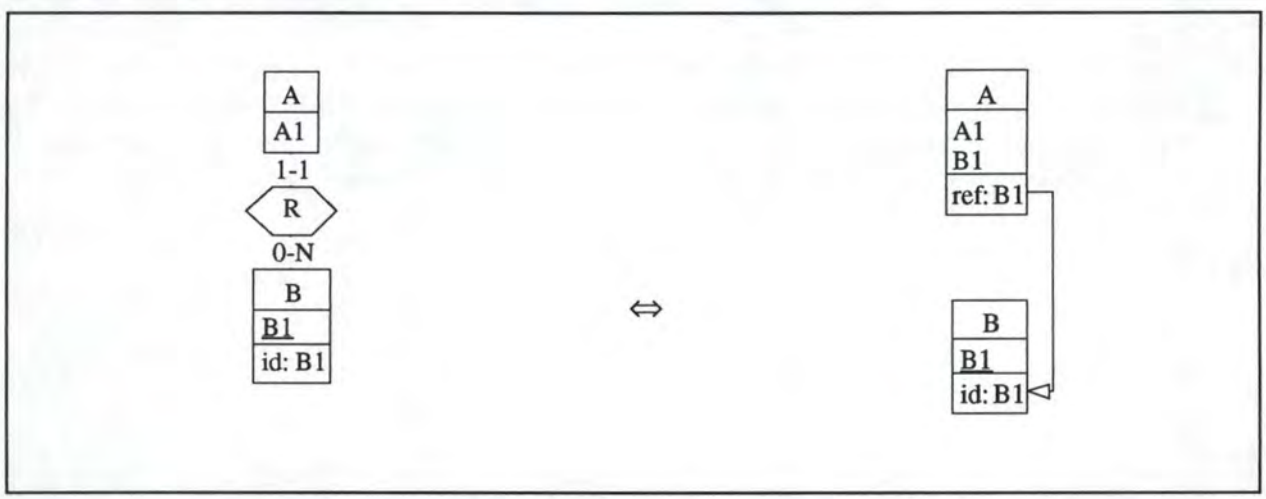


Figure 7: Transformation d'un T.A. one-to-many en clés étrangères.

1.6 *Transformation de rôle multi-type d'entités*

Un rôle multi-T.E. est éliminé en scindant son type d'associations R de manière à le distribuer entre les T.E. sur lesquels il est défini comme présenté à la Figure 8. Cela mène à des structures faciles à traduire en composants conformes au modèle relationnel.

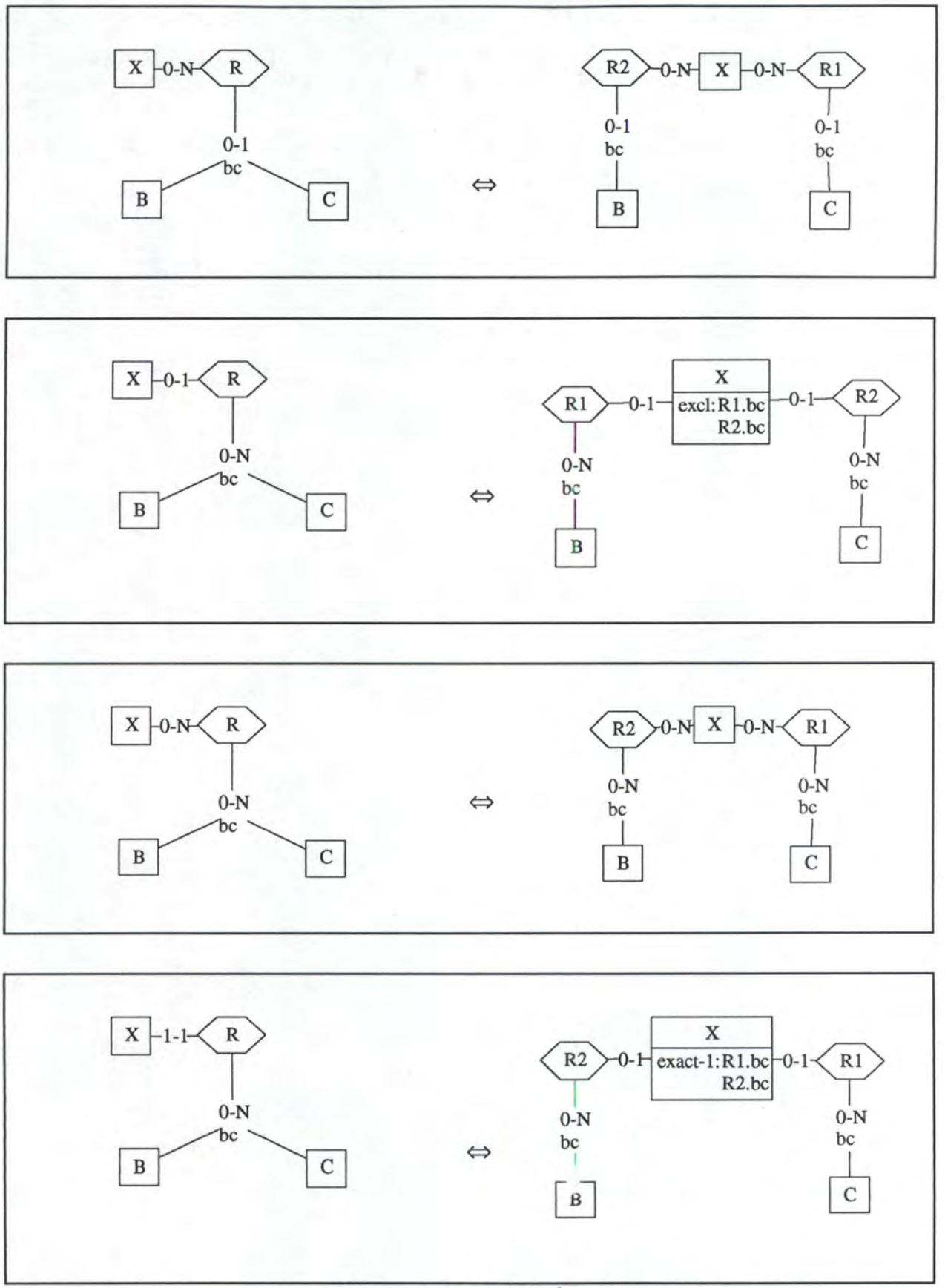


Figure 8: Distribution d'un rôle multi-T.E.

1.7 Transformation de type d'associations many-to-many

Un T.A. many-to-many peut se transformer en un T.E. Deux T.A. one-to-many sont créés. Ils peuvent à leur tour être transformés (paragraphe 1.1). La Figure 9 montre la transformation d'un tel T.A.

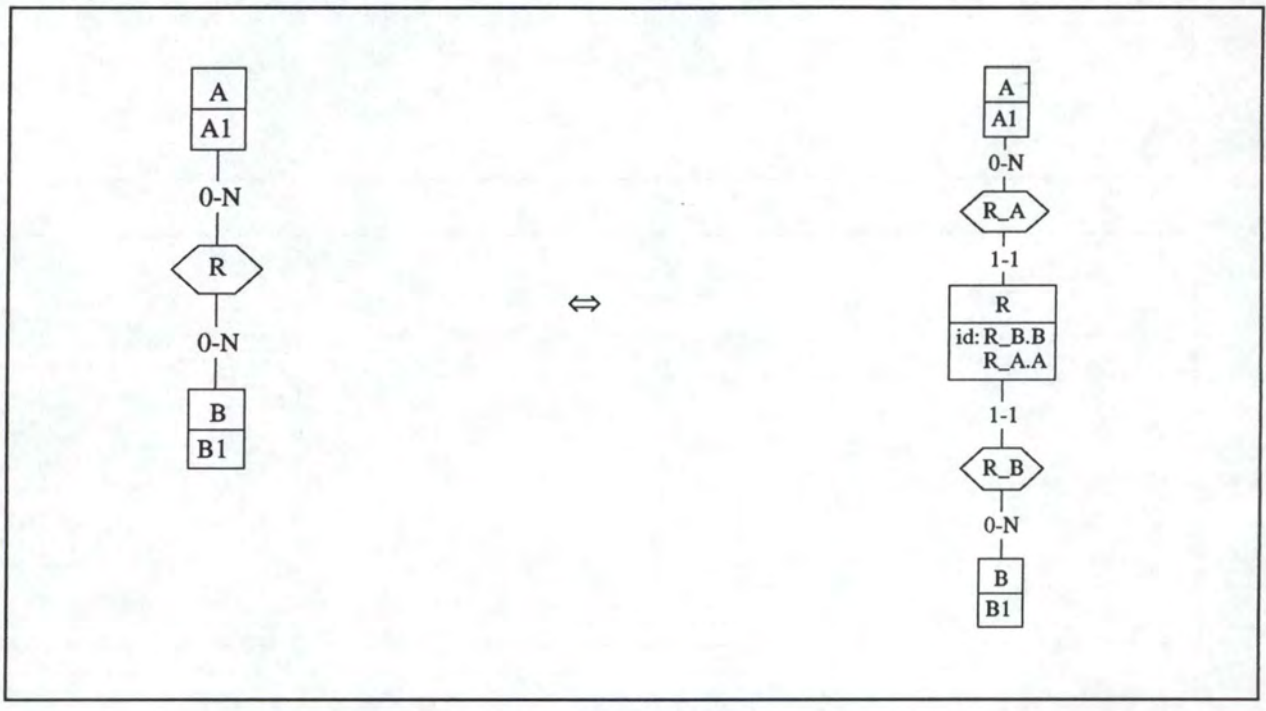


Figure 9 : Transformation d'un T.A. many-to-many

2. *Analyse préliminaire*[10]

Avant de commencer une analyse plus détaillée des relations is-a, il est bon d'éclaircir la situation.

Les relations is-a que nous allons tout d'abord étudier ne concernent que l'héritage simple, c'est-à-dire où un sous-type n'est impliqué que dans une seule relation is-a.

Types de relations is-a

- Dans le cas de la transformation par matérialisation, l'élément qui permet de définir les différents types de relations is-a est la présence d'identifiants dans le surtype et/ou les sous-types.

  Trois types de relations is-a ont été choisies pour montrer que les conditions nécessaires pour avoir une transformation par matérialisation avec transformation des T.A. en clés étrangères sont la présence d'identifiant dans le surtype ou la présence d'identifiant dans tous les sous-types.

  S'il n'y a pas d'identifiant dans le surtype ou que les sous-types n'ont pas tous d'identifiant, le paragraphe 1.1 nous permet de déduire qu'il n'y aura pas de transformation des T.A. en clés étrangères. La transformation par matérialisation n'est alors pas possible.

- La présence d'un T.A. lié à un sous-type définit un type supplémentaire de relations is-a.
- Le fait que le surtype joue un rôle dans un T.A. définit un dernier type de relations is-a.

Les relations is-a étudiées sont soit soumises à des contraintes de disjonction, de totalité et de partition ou ne sont soumises à aucune contrainte.

L'étude qui va suivre est divisée en paragraphes selon le type de relation étudié et pour illustrer ces différents types de relations is-a, nous avons repris des exemples de relations is-a avec deux sous-types.

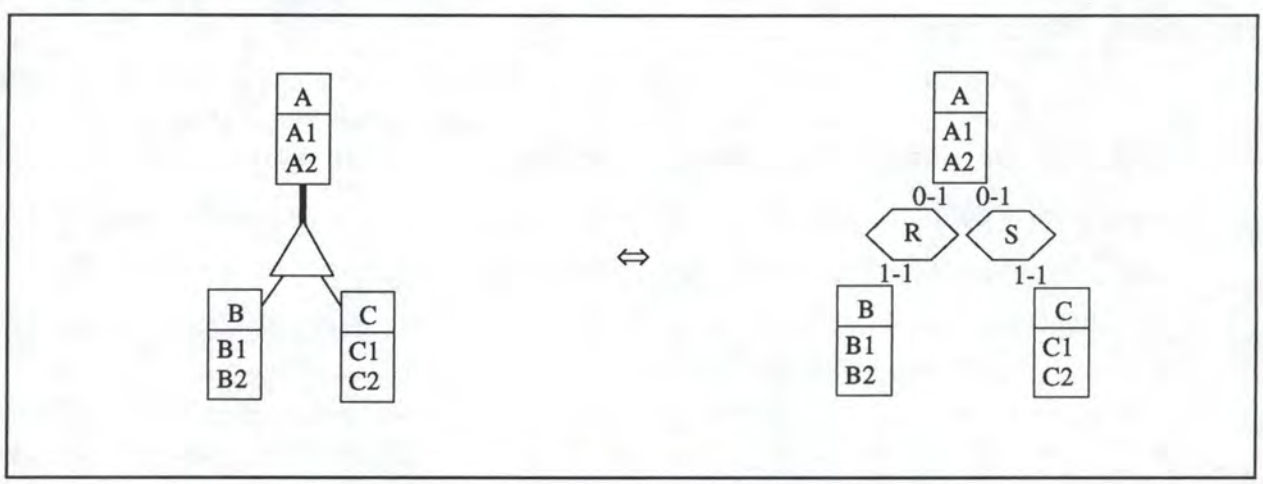
3. *Etude des relations is-a transformées par la technique de matérialisation* [1],[3],[4],[7],[8],[10]

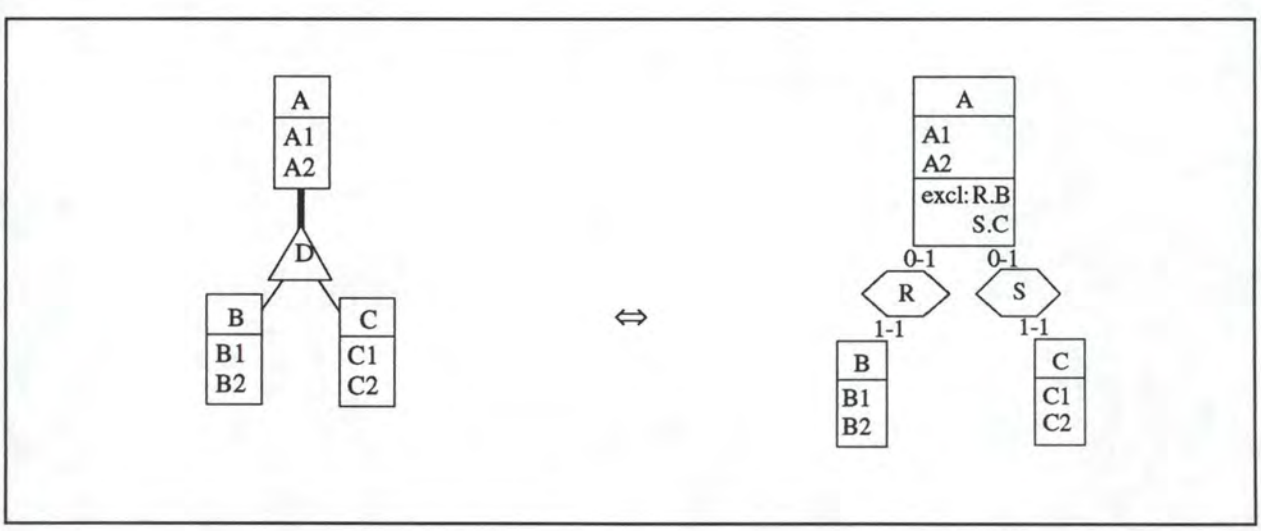
La transformation de relations is-a par matérialisation introduit des types d'associations (T.A.) qui vont être transformés en clés étrangères. Pour permettre la transformation par matérialisation avec création de clés étrangères, il est nécessaire d'avoir un identifiant dans le surtype ou des identifiants dans tous les sous-types.

3.1 *Relation is-a sans identifiant dans le surtype, ni dans les sous-types*

3.1.1 *Relations intéressantes et leurs transformations*

Par la transformation de matérialisation is-a, la relation is-a est remplacée par des T.A. one-to-one (Figure 10).





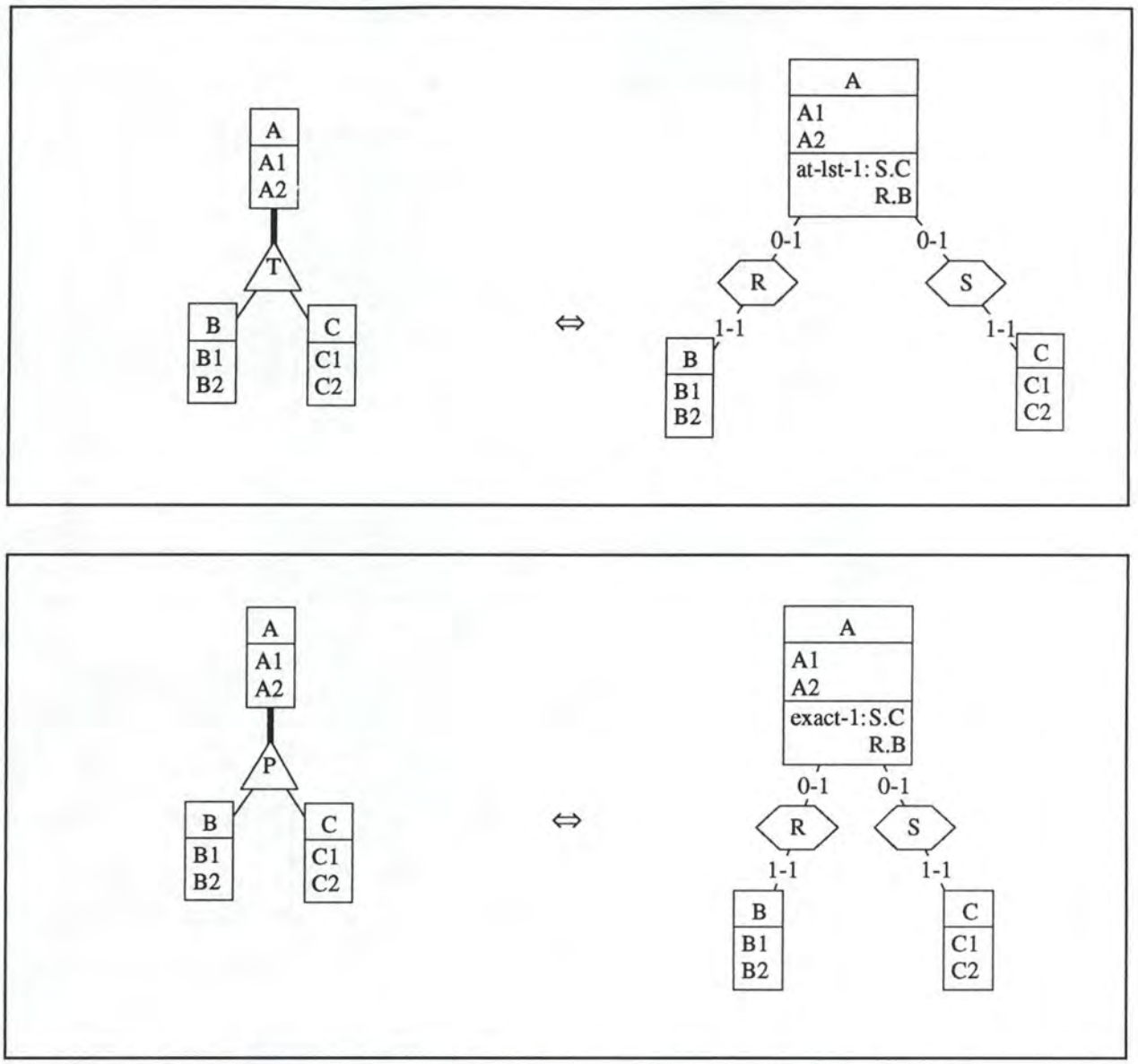


Figure 10: Transformation par matérialisation (relation is-a sans identifiant dans surtype ni dans sous-types)

Pour que les schémas soient relationnels, il est nécessaire de transformer les T.A. one-to-one.

Comme le surtype ne possède pas d'identifiant, les T.A. ne peuvent pas être transformés en clés étrangères. Si l'on désire un schéma conforme au modèle relationnel, on peut utiliser les transformations vues au paragraphe 1.1.

3.2 *Relation is-a ayant un identifiant dans le surtype uniquement*

Lors de la transformation de matérialisation is-a, de T.A. one-to-one sont créés. La transformation de ces T.A. se fait en ajoutant aux sous-types une clé étrangère référençant le surtype. Pour pouvoir exprimer les contraintes de disjonction ou de totalité, on peut ajouter au T.E. correspondant au surtype des attributs booléens qui attestent de la présence ou non des T.E. sous-types. En effet, un tel attribut n'est pas nul si le surtype appartient au sous-type correspondant. Les contraintes de disjonction et de totalité sont exprimées sur ces attributs.

La Figure 11 montre les résultats d'une telle transformation.

3.2.1 *Relations intéressantes et leurs transformations*

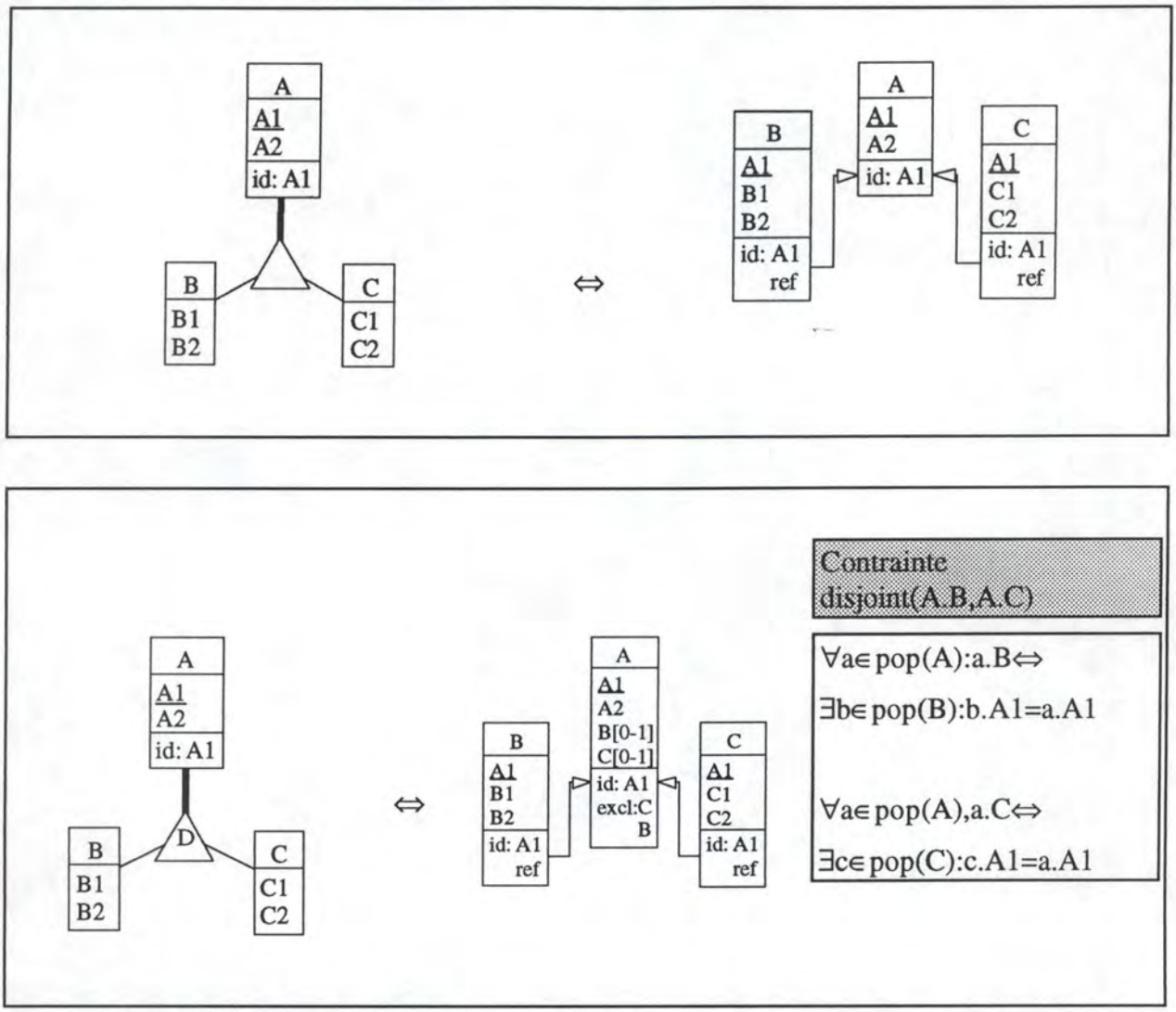


Figure 11: Transformation par matérialisation (relation is-a avec identifiant dans surtype)

3.2.2 *Code SQL*

Lors de cette transformation, des attributs booléens sont créés dans le surtype. Le code SQL pour créer la table correspondant au surtype est :

```
create table A (
      A1 not null,
      A2 not null,
      B char(1),
      C char(1),
primary key(A1));
```

Le code SQL qui traduit la contrainte de disjonction entre les deux attributs booléens est :

```
alter table A add constraint ISAA
check ((C is not null and B is null)
    or (C is null and B is not null)
    or (C is null and B is null));
```

Pour la contrainte de totalité entre les deux attributs, le code SQL est :

```
alter table A add constraint ISAA
check (C is not null or B is not null);
```

Pour la contrainte de partition, le code SQL est :

```
alter table A add constraint ISAA
check ((C is not null and B is null)
    or (C is null and B is not null));
```

Le code SQL correspondant à la contrainte qui vérifie la présence de valeur dans les attributs si le surtype appartient à un sous-type, est le suivant :

```
alter table B add constraint CHK_B
check(not exists
      (select B from A where A.A1=B.A1 and B is null)) ;

alter table C add constraint CHK_C
check(not exists
      (select C from A where A.A1=C.A1 and C is null)) ;

alter table ...

alter table Z add constraint CHK_Z
check(not exists
      (select Z from A where A.A1=Z.A1 and Z is null)) ;
```

Une remarque est cependant à formuler : certains SGBD n'acceptent pas de checks avec une clause qui mentionnent d'autres tables.

3.2.3 Elimination des contraintes

Une alternative à cette méthode est de changer la contrainte *excl, at-lst-1* ou *exact-1* exprimée dans le surtype, en check qui vérifie la disjonction ou/et la totalité entre les sous-types (voir Figure 4). Cette méthode sera apppelée dans la suite de l'exposé, la transformation par matérialisation simple.

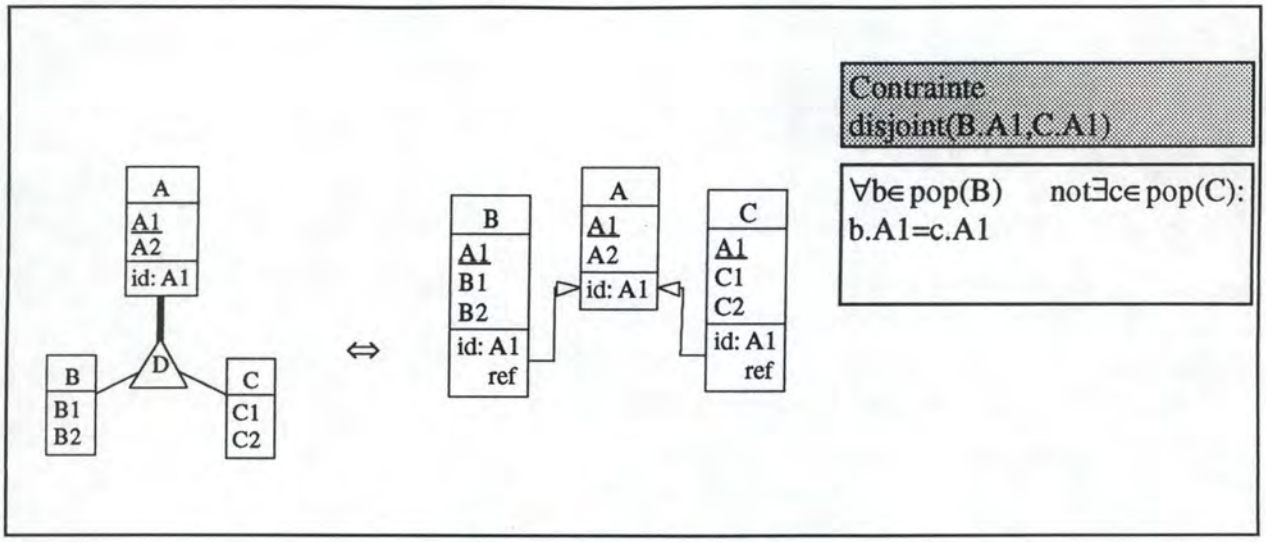


Figure 12 : Elimination de contraintes

Pour une contrainte de totalité, nous pouvons nous reporter à la Figure 4.

3.2.4 Code SQL

Le code SQL du check correspondant à la contrainte disjoint(B.A1,C.A1) est :

```
alter table B add constraint DIS_B
check(A1 not in
      (select A1 from C)
  and ...
  and (A1 not in
      (select A1 from Z)) ;

alter table C add constraint DIS_C
check(A1 not in
      (select A1 from B)
  and ...
  and (A1 not in
      (select A1 from Z)) ;
...
alter table Z add constraint DIS_Z
check(A1 not in
      (select A1 from B)
and (A1 not in
      (select A1 from C)
and ...) ;
```

Remarquons qu'il est nécessaire d'avoir autant de checks que de sous-types pour pouvoir exprimer correctement la contrainte de disjonction.

Pour la contrainte de totalité, le code est :

```
alter table A add constraint TOT_A
check(A1 in
      (select A1 from B)
   or (A1 in
      (select A1 from C)
   or...
   or (A1 in
      (select A1 from Z))) ;
```

Pour N sous-types, (N*N-1)/2 checks sont générés pour coder la contrainte de disjonction. Ce nombre quadratique croissant de checks nous pousse à utiliser une alternative : l'indicateur de sous-type.

3.2.5 Elimination des contraintes par la méthode de l'indicateur de sous-type

Une méthode utilisée par certains auteurs est celle de l'indicateur de sous-types. Lors de la transformation de la relation is-a par matérialisation, des T.A. one-to-one sont créés. Lors de la transformation de ces T.A. en clés étrangères, un attribut TYPE est ajouté au T.E. surtype. Cet attribut indique à quels sous-types chaque surtype appartient (Figure 13).

La cardinalité de cet attribut traduit les contraintes du surtype comme suit (N est le nombre de sous-types) :

disjoint	→ TYPE[0-1]
total	→ TYPE[1-N]
partition	→ TYPE[1-1]
pas de contrainte	→ TYPE[0-N]

Cet attribut contient le nom des sous-types auxquels le surtype appartient. Son domaine est donc, dans le cas de la Figure 13 : {"B"}∪{"C"}∪{"B","C"}

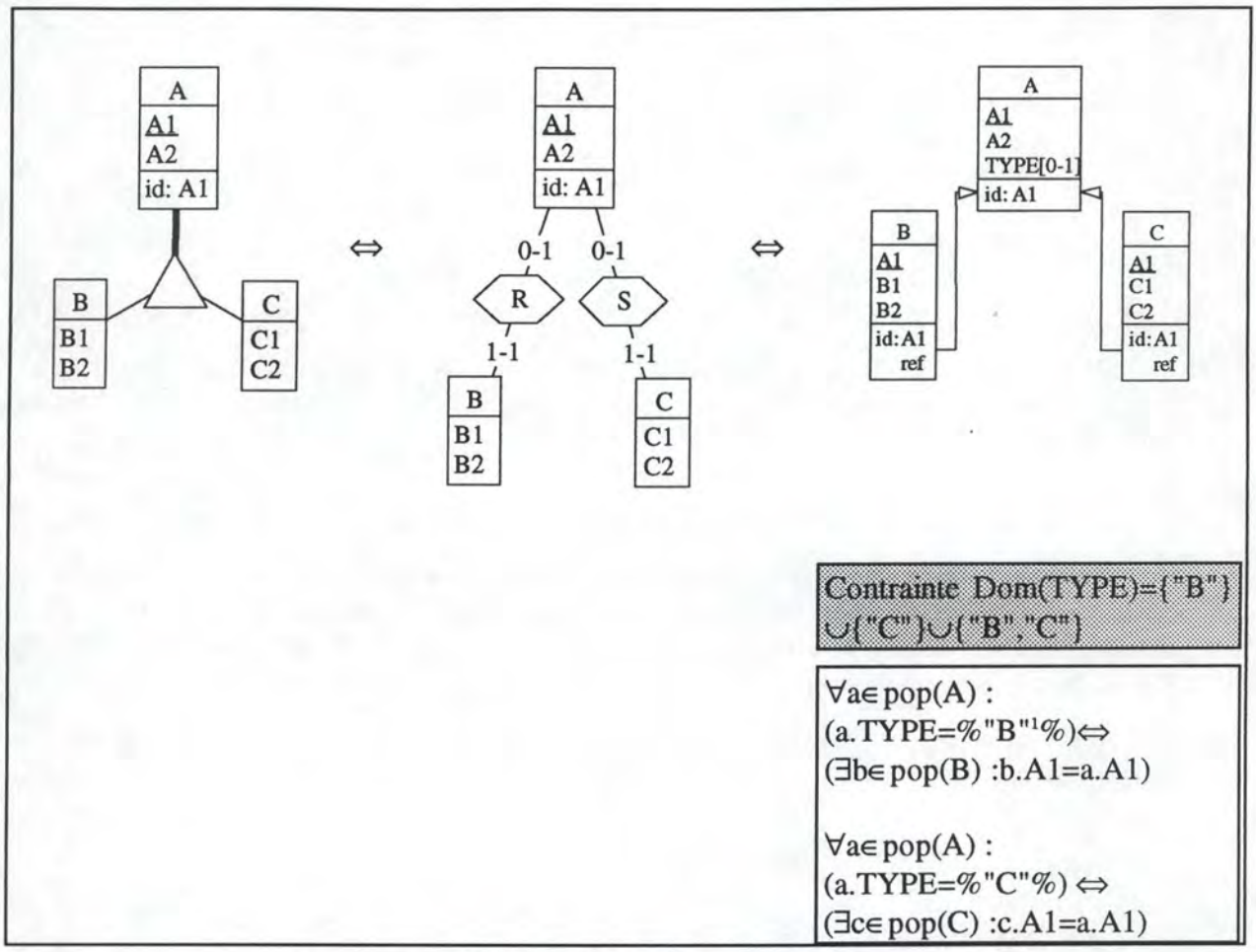
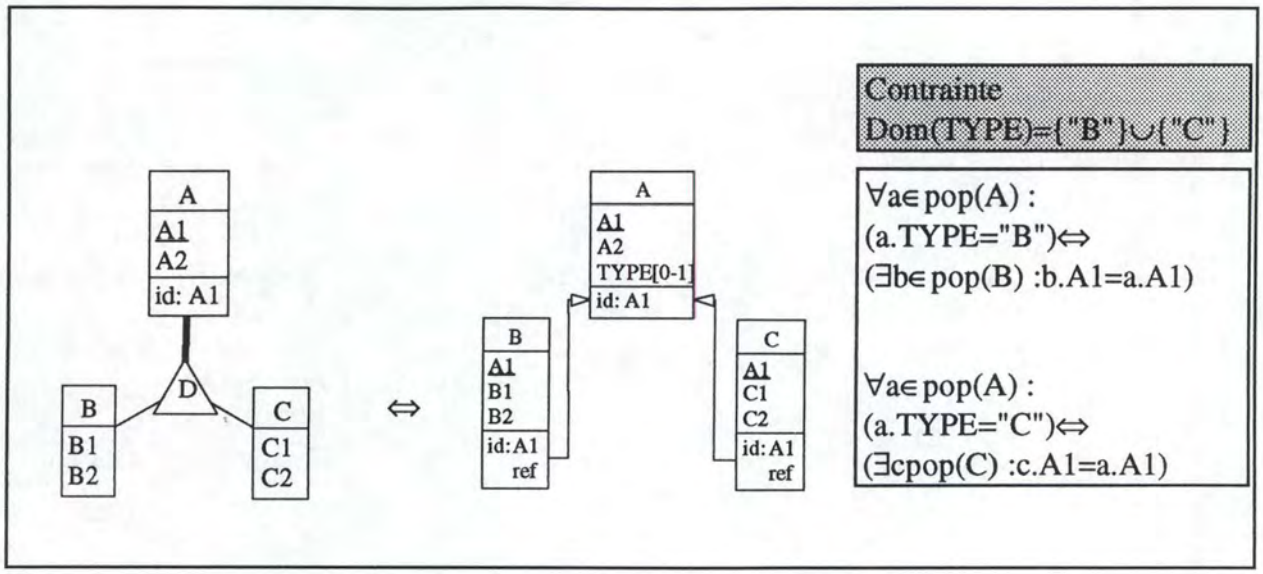
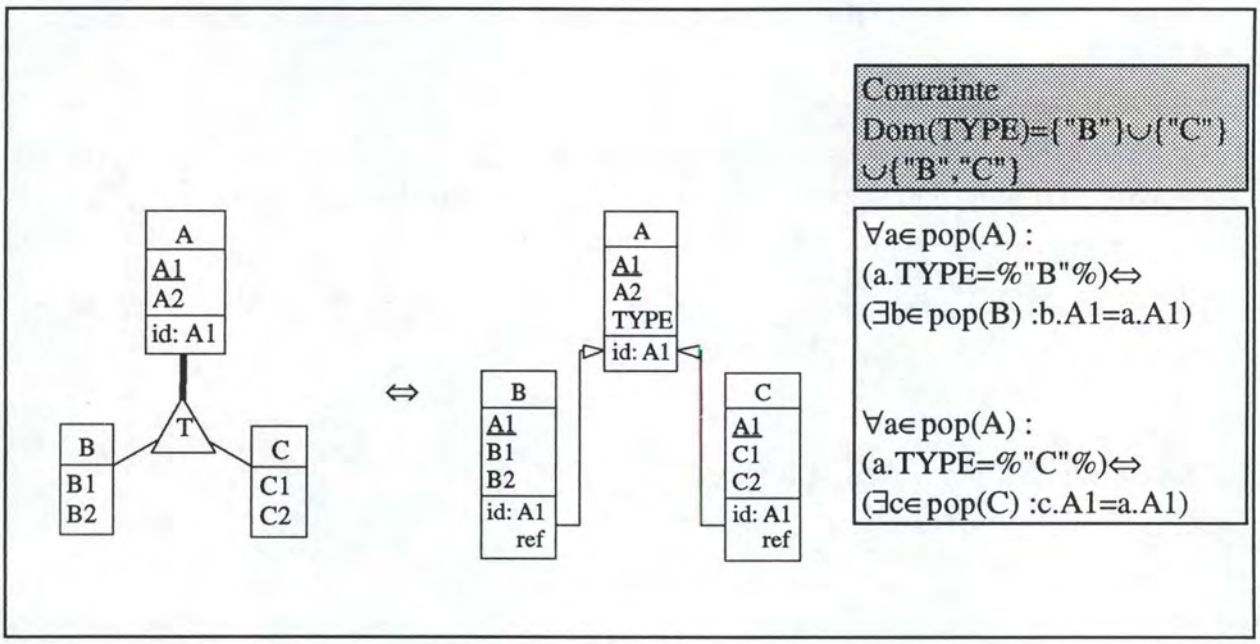


Figure 13 : Méthode de l'indicateur de sous-types

Pour les relations is-a ayant une contrainte de disjonction, de totalité ou de partition, les contraintes sont identiques, seule la cardinalité de l'attribut TYPE change (Figure 14). Pour la contrainte de disjonction, un attribut TYPE multivalué est créé. Il doit être transformé en un attribut monovalué, TYPES. La valeur de TYPES est celle de la concaténation des valeurs de l'ancien attribut TYPE. Le choix de transformation de l'attribut TYPE est motivé par la facilité de consultation des valeurs de cet attribut. Par simplification, nous avons considéré que l'attribut multivalué et monovalué ont le même nom : TYPE.

[1] Cette représentation indique que l'attribut TYPE contient "B"


A
A1
A2
id: A1
D
B
B1
B2
C
C1
C2
⇔
A
A1
A2
TYPE[0-1]
id: A1
B
A1
B1
B2
id:A1
ref
C
A1
C1
C2
id:A1
ref
Contrainte
Dom(TYPE)={"B"}∪{"C"}
∀a∈ pop(A) :
(a.TYPE="B")⇔
(∃b∈ pop(B) :b.A1=a.A1)
∀a∈ pop(A) :
(a.TYPE="C")⇔
(∃cpop(C) :c.A1=a.A1)


A
A1
A2
id: A1
T
B
B1
B2
C
C1
C2
⇔
A
A1
A2
TYPE
id: A1
B
A1
B1
B2
id: A1
ref
C
A1
C1
C2
id: A1
ref
Contrainte
Dom(TYPE)={"B"}∪{"C"}
∪{"B","C"}
∀a∈ pop(A) :
(a.TYPE=%"B"%)⇔
(∃b∈ pop(B) :b.A1=a.A1)
∀a∈ pop(A) :
(a.TYPE=%"C"%)⇔
(∃c∈ pop(C) :c.A1=a.A1)

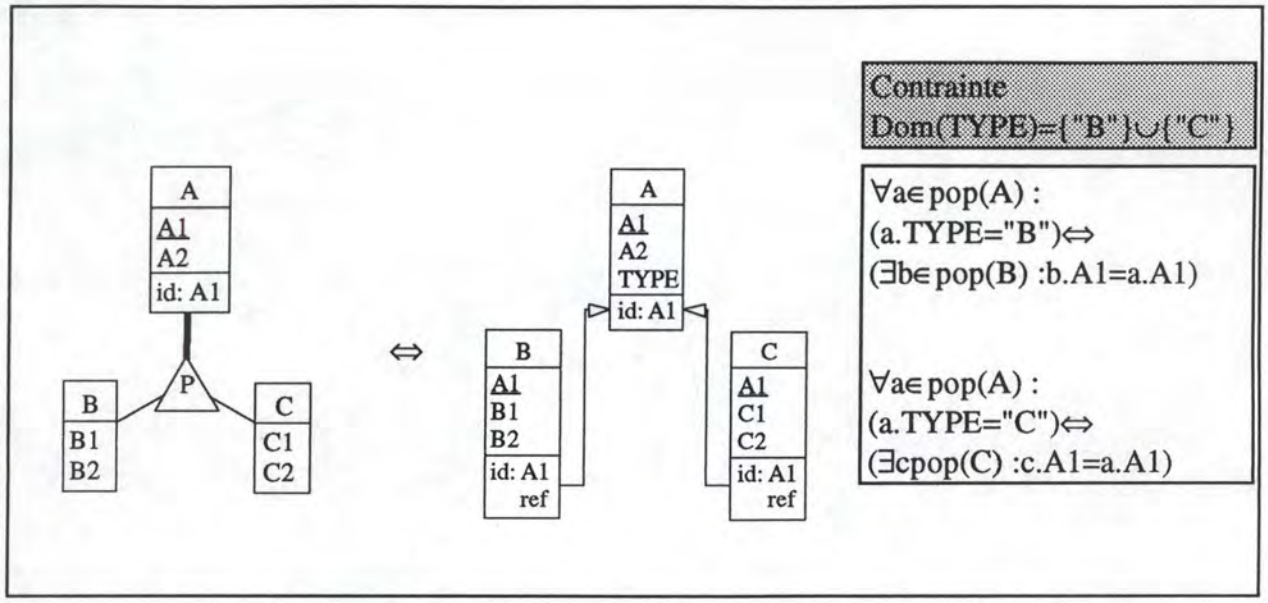


Figure 14 : Transformation par la méthode de l'indicateur de sous-type

3.2.6 Code SQL

La transformation par création d'un indicateur de sous-types place un attribut TYPE dans le surtype. Le code SQL de création de la table correspondant au surtype en devient :

```
create table A (
    A1 char(1) not null,
    A2 char(1) not null,
    TYPE varchar(X) (not null)[2],
    primary key(A1)) ;
```

La contrainte s'exprime en SQL par :

```
alter table B add constraint TYPE_B
check (not exists
        (select TYPE from A
         where A.A1 = B.A1
         and TYPE not like '%"B"%' ));

alter table C add constraint TYPE_C
check (not exists
        (select TYPE from A
         where A.A1 = C.A1
         and TYPE no like '%"C"%' ));

...
```

[2] la présence du not null dépend de la contrainte sur le surtype

```
alter table Z add constraint TYPE_Z
check (not exists
        (select TYPE from A
         where A.A1 = Z.A1
         and TYPE not like '%"Z"%'));
```

Remarquons que l'on aurait pu choisir de remplir l'attribut TYPE avec les identifiants des sous-types auxquels le surtype appartient.

Une autre alternative à cette méthode est d'utiliser un compteur de sous-type, qui dans l'attribut TYPE met le nombre de sous-types auxquels le surtype appartient. Une série de checks est alors nécessaire pour vérifier les contraintes de disjonction (nombre = 1 ou 0), de totalité (nombre ≤ 1) ou de partition (nombre = 1).

3.2.7 Conclusion

Dans ce paragraphe, trois méthodes de transformation ont été abordées :

- la méthode de transformation de matérialisation simple où les contraintes sont exprimées sur les T.E. sous-types
- la méthode de transformation avec créations d'attributs booléens dans le surtype où les contraintes sont exprimées sur les attributs booléens
- la méthode de l'indicateur de sous-types.

Les avantages et les inconvénients de chacune des méthodes sont présentés dans le Tableau 1.

	Avantages	**Inconvénients**
Contraintes sur attributs booléens	Contraintes représentées dans le schéma	Utilise transfo. non réversible checks et trigger nécessaires
Contraintes sur T.E. sous-types	Moins de checks que méthode précédente	Checks pas acceptés par tous les SGBD
Indicateur de sous-types	Permet de retrouver à partir du surtype les sous-types auquel il appartient	Utilise transfo. non réversible checks et trigger nécessaires

Tableau 1 : Avantages et inconvénients des méthodes

3.3 Relation is-a n'ayant pas d'identifiant dans tous les sous-types et n'ayant pas d'identifiant dans le surtype

La transformation de la relation is-a par matérialisation donne deux T.A. one-to-one, comme vu précédemment.

La transformation des T.A. one-to-one ne peut se faire par création de clés étrangères car le surtype ne possède pas d'identifiant.

Si l'on veut transformer les T.A. one-to-one en clés étrangères, il faut ajouter un identifiant technique au surtype et cela revient à l'étude du paragraphe 3.2.

3.4 *Relation is-a où le surtype joue un rôle dans un T.A.*

Dans une relation is-a où le surtype joue un rôle, le T.A. dans lequel le surtype joue un rôle doit être transformé. Deux cas intéressants de transformation de T.A. en clés étrangères peuvent être observés :

- le surtype est origine de la clé étrangère
- le surtype est la destination de la clé étrangère.

Ces transformations nécessitent un identifiant dans le type d'entités X ou dans le surtype.

Les schémas correspondant au résultat de ces transformations sont présentés à la Figure 15.

3.4.1 *Relations intéressantes et leurs transformations*

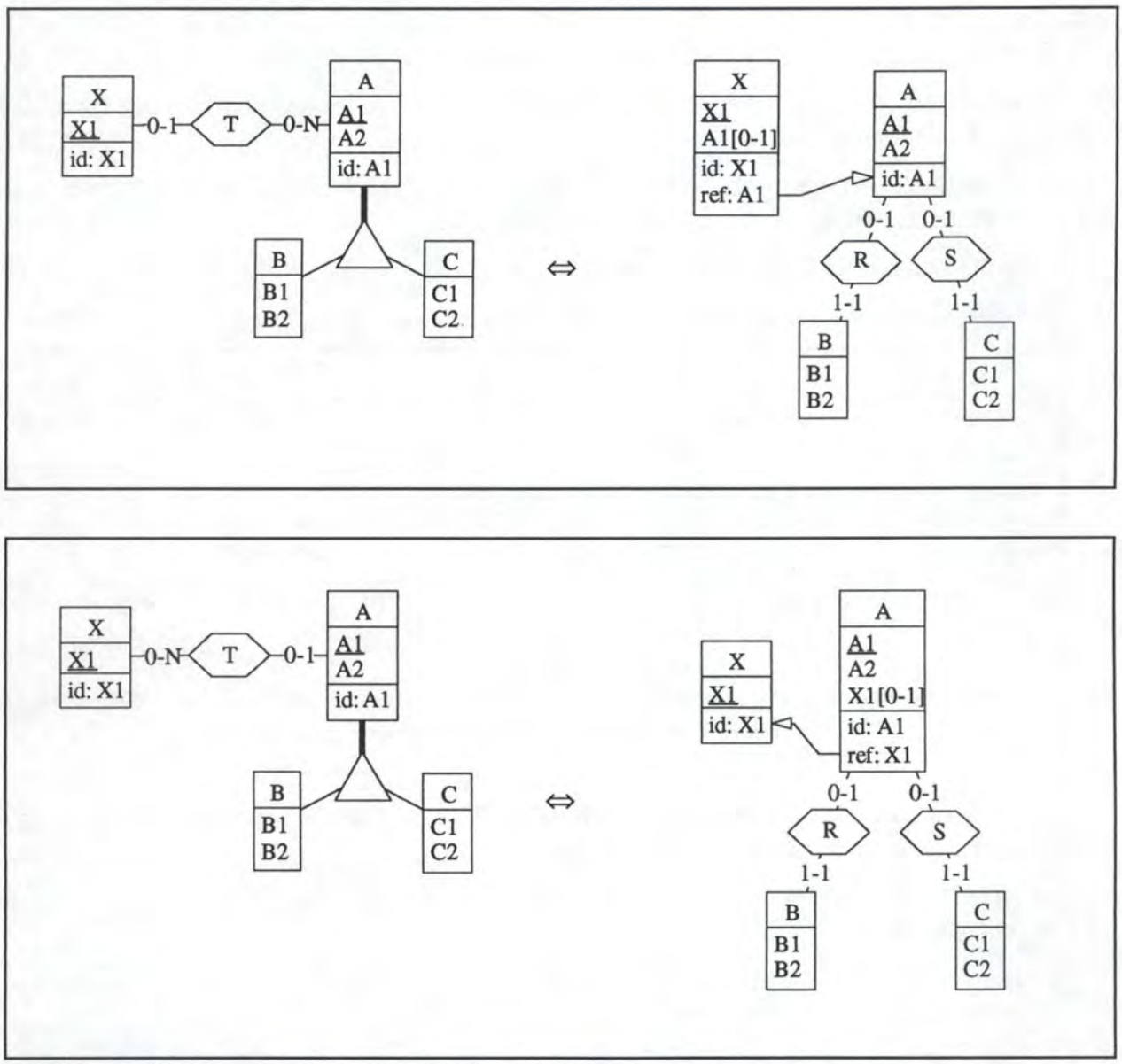


Figure 15 : Transformation par matérialisation de relation is-a (surtype joue un rôle)

3.4.2 *Code SQL*

Les deux T.A. one-to-one sont traduits en clés étrangères et le code SQL de ces schémas ne pose aucun problème particulier.

3.5 *Relation is-a où un sous-type au moins joue un rôle dans un T.A.*

On peut transformer en clés étrangères les T.A. one-to-one générés lors de la transformation de la relation is-a où un sous-type joue un rôle. Cette transformation ne pose aucun problème particulier.

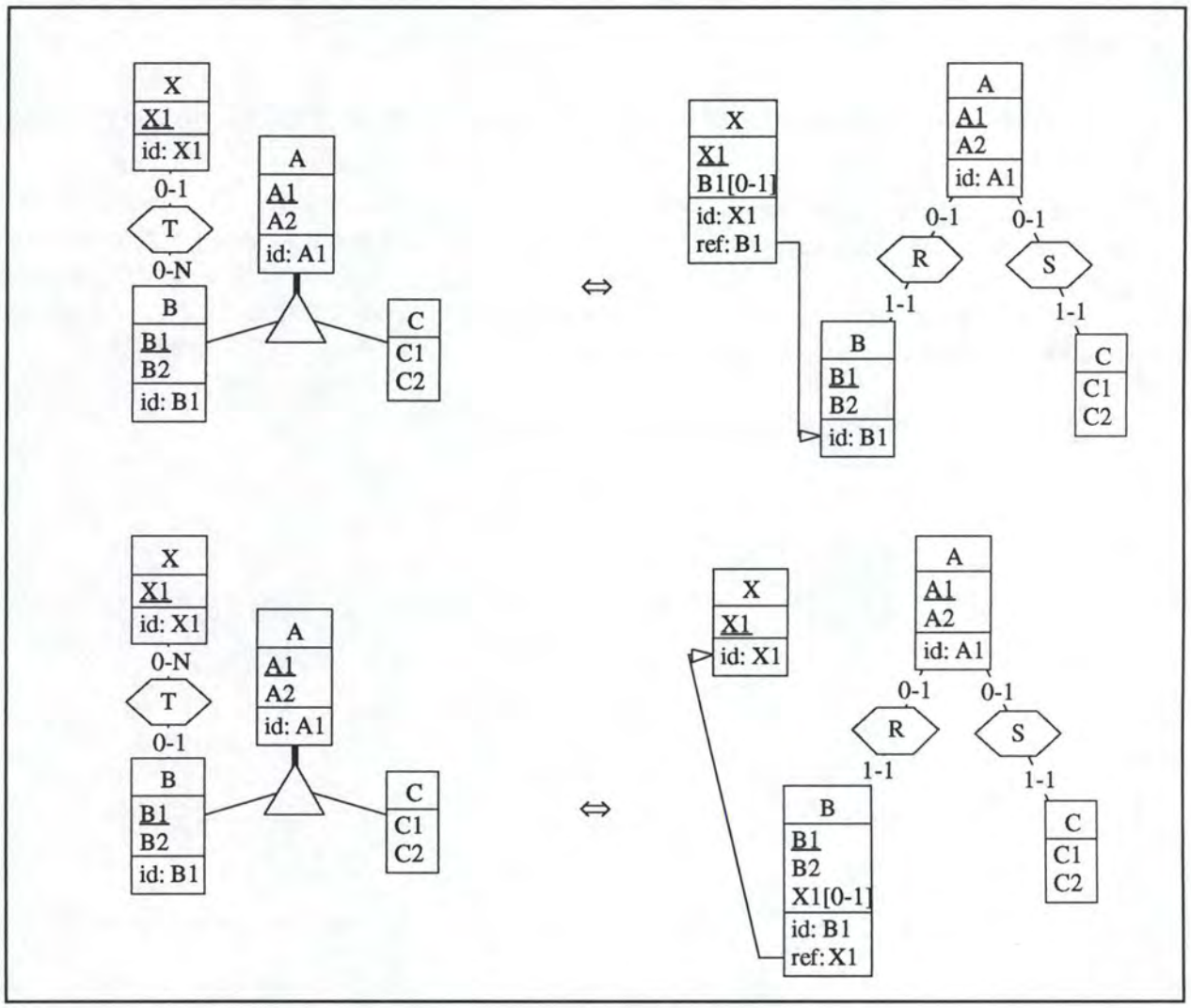


Figure 16 : Transformation par matérialisation de relation is-a (sous-type joue un rôle)

4. *Etude des relations is-a, transformées par la technique d'héritage ascendant* [7],[8],[10]

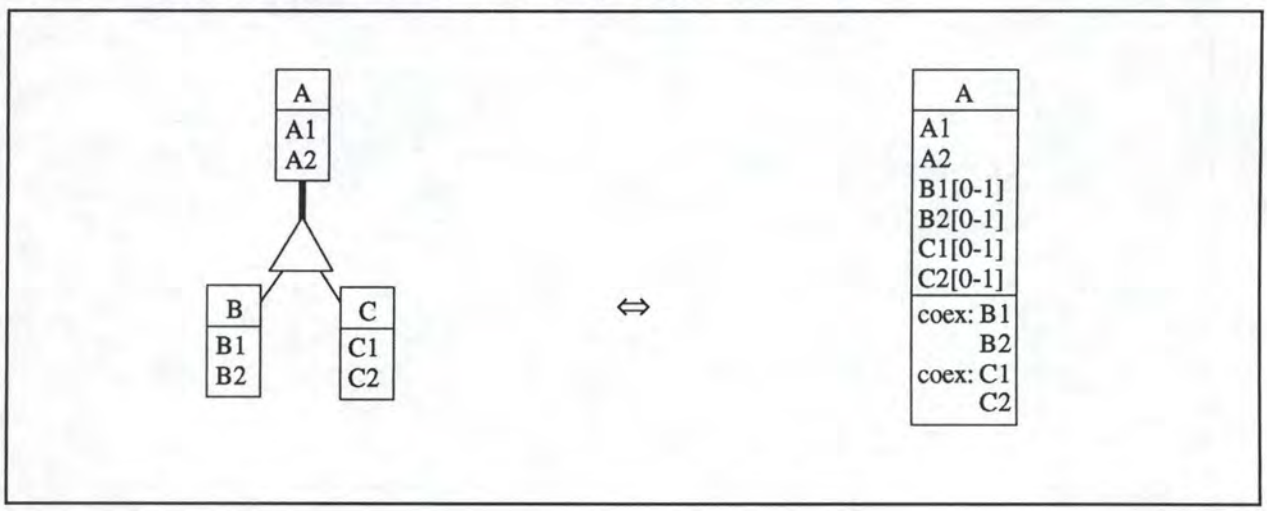
Par la transformation d'héritage ascendant, il y a représentation du surtype. Les sous-types sont fusionnés dans le surtype.

Pour permettre la fusion des sous-types dans le surtype, aucune précondition n'est nécessaire et seules les relations is-a dont le sous-type joue un rôle dans un T.A. doivent faire l'objet d'une transformation un peu particulière : il y a migration du rôle que joue le sous-type vers le surtype.

4.1 *Relation is-a sans identifiant dans le surtype, ni dans les sous-types*

La transformation par héritage ascendant inclut des contraintes de coexistence entre les attributs provenant des sous-types. Comme les contraintes de disjonction, de totalité et de partition portent sur les ensembles {B1 B2} et {C1 C2} et de par la contrainte de coexistence entre B1 et B2 et celle entre C1 et C2, on peut simplifier les contraintes de disjonction, de totalité et de partition en excl : B1,C1, at-lst-1 : B1,C1, exact-1 : B1,C1 (Figure 17).

4.1.1 *Relations intéressantes et leurs transformations*



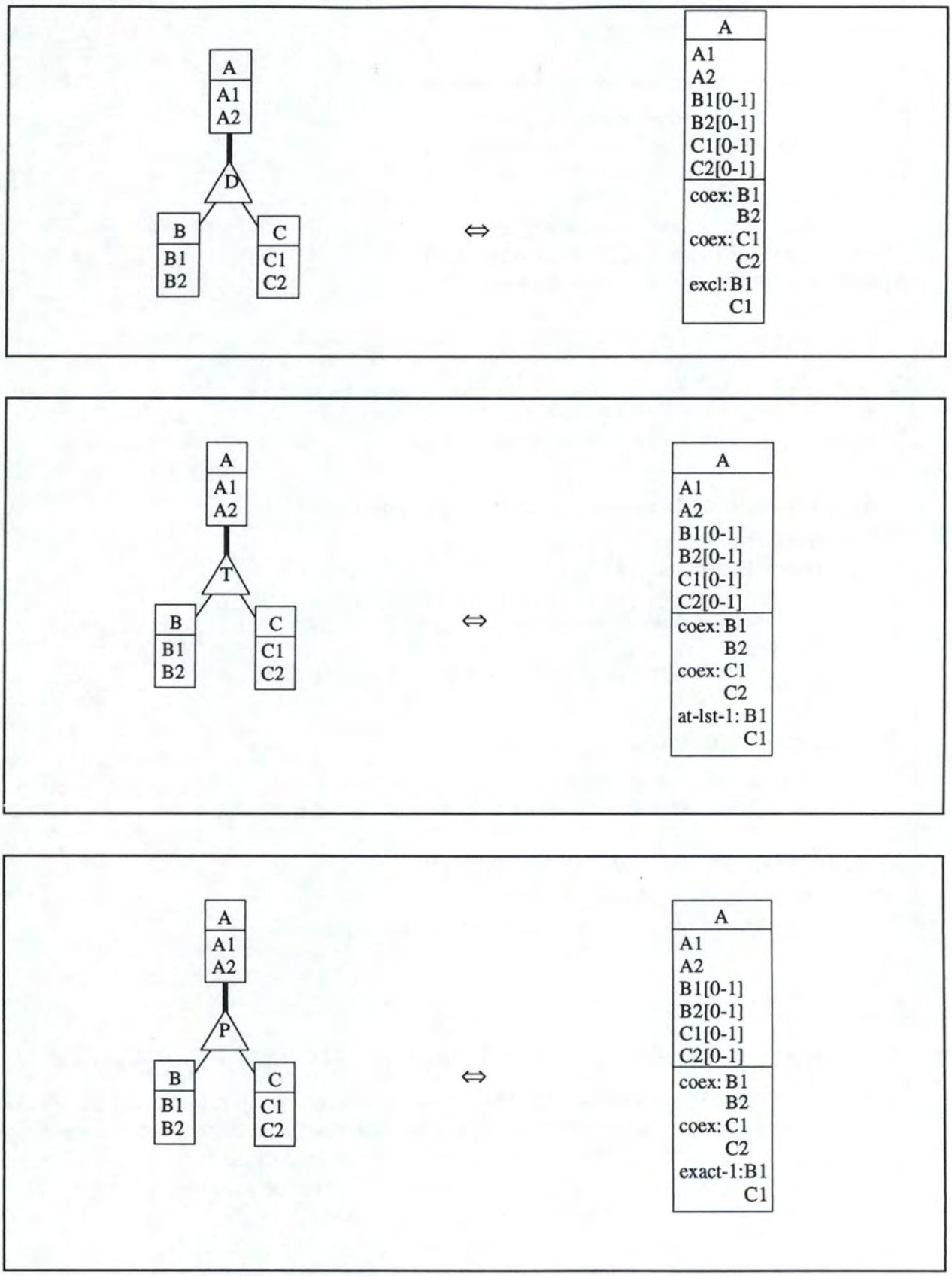


Figure 17 : Transformation par héritage ascendant

4.1.2 *Code SQL*

Les contraintes de coexistence se traduisent par :

```
alter table A add constraint COEXA
check((B1 is not null and B2 is not null)
    or (B1 is null and B2 is null));

alter table A add constraint COEXA_1
check((C1 is not null and C2 is not null)
    or (C1 is null and C2 is null));

...

alter table A add constraint COEXA_2
check((Z1 is not null and Z2 is not null)
    or (Z1 is null and Z2 is null));
```

Pour la contrainte de disjonction, le code correspondant est :

```
alter table A add constraint ISAA
check((C1 is not null and B1 is null and...and Z1 is null)
    or (C1 is null and B1 is not null and...and Z1 is null)
    or (C1 is null and B1 is null and...and Z1 is null)
    or...
    or (C1 is null and B1 is null and...and Z1 is not null));
```

Pour la contrainte de totalité, le code généré est :

```
alter table A add constraint ISAA
check(C1 is not null or B1 is not null or...orZ1 is not null);
```

Pour la contrainte de partition, le code généré est :

```
alter table A add constraint ISAA
check((C1 is not null and B1 is null and...and Z1 is null)
    or (C1 is null and B1 is not null and ...and Z1 is null)
    or...
    or (C1 is null and B1 is null and...and Z1 is not null));
```

4.2 *Relation is-a ayant un identifiant dans le surtype uniquement*

Dans les relations is-a ayant un identifiant dans le surtype, la transformation par héritage ascendant est possible et les résultats obtenus sont identiques à ceux présentés au paragraphe précédent. La Figure 18 reprend les résultats de la transformation par héritage ascendant pour la relation is-a n'ayant pas de contrainte. Les transformations de relations is-a possédant une contrainte dans le surtype peuvent être déduites de la Figure 17.

4.2.1 *Relations intéressantes et leurs transformations*

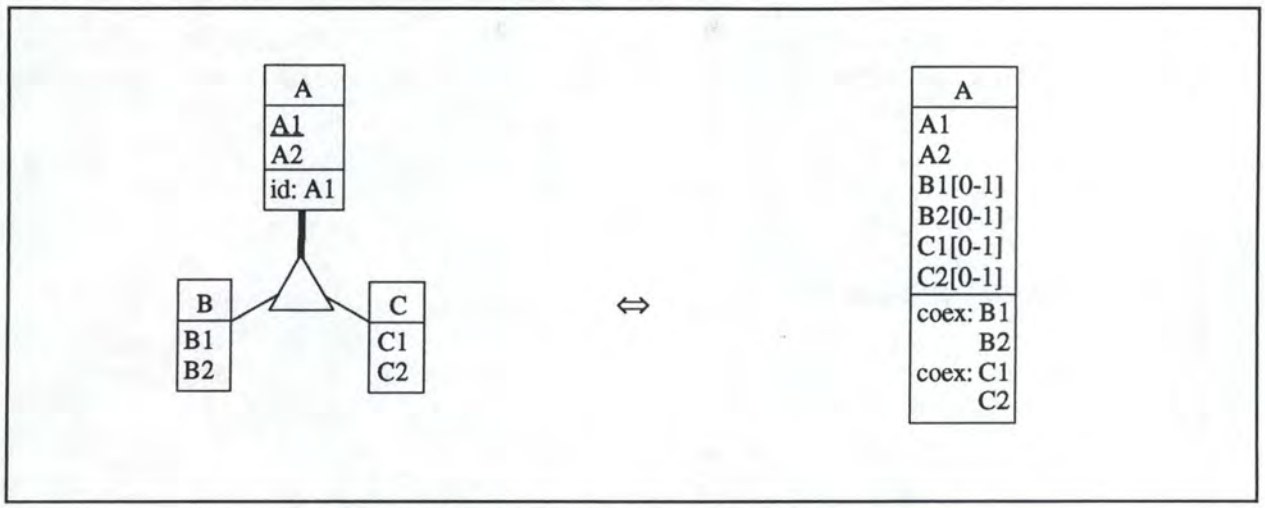


Figure 18 : Transformation par héritage ascendant

De même le code SQL reprend les contraintes de disjonction, de totalité et de partition du paragraphe précédent.

4.3 *Relation is-a n'ayant pas d'identifiant dans tous les sous-types et n'ayant pas d'identifiant dans le surtype*

La Figure 19 reprend la transformation par héritage ascendant pour la relation is-a n'ayant pas de contrainte. Cette transformation ne pose pas de problème particulier : les résultats sont identiques à ceux présentés au paragraphe 4.1.

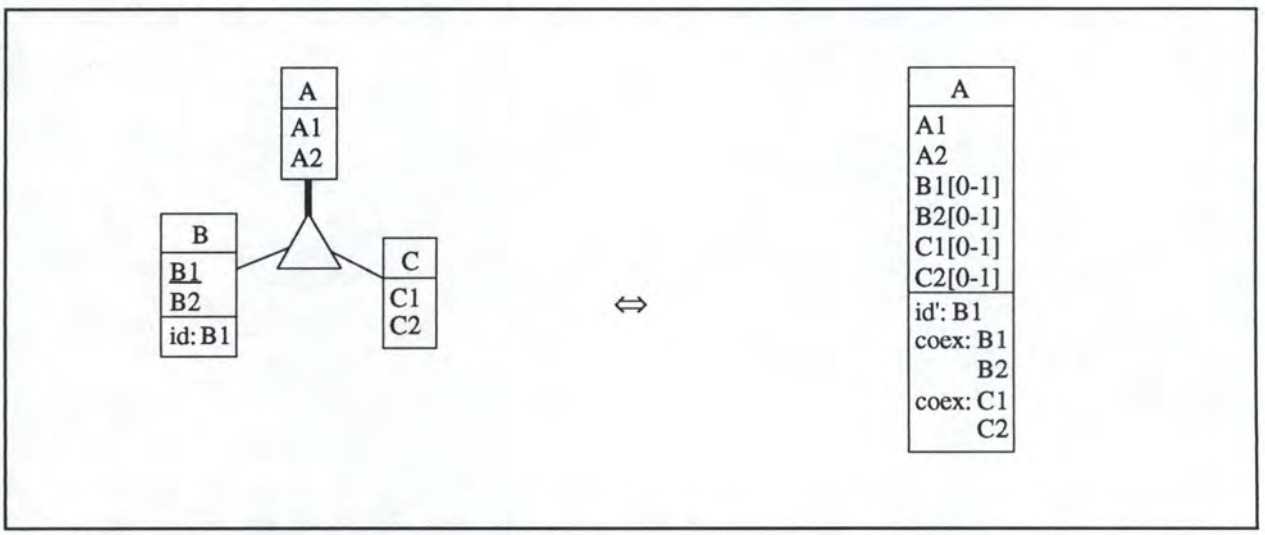


Figure 19 : Transformation par héritage ascendant de relation is-a (sous-type a un identifiant)

4.4 *Relation is-a où le surtype joue un rôle dans un T.A.*

Pour la relation is-a où le surtype joue un rôle dans un T.A., la transformation ne présente pas de caractéristique particulière : les résultats obtenus après transformation sont présentés à la Figure 20. Pour les autres types de T.A.(relations one-to-one, many-to-one, many-to-many), les résultats peuvent être facilement déduits.

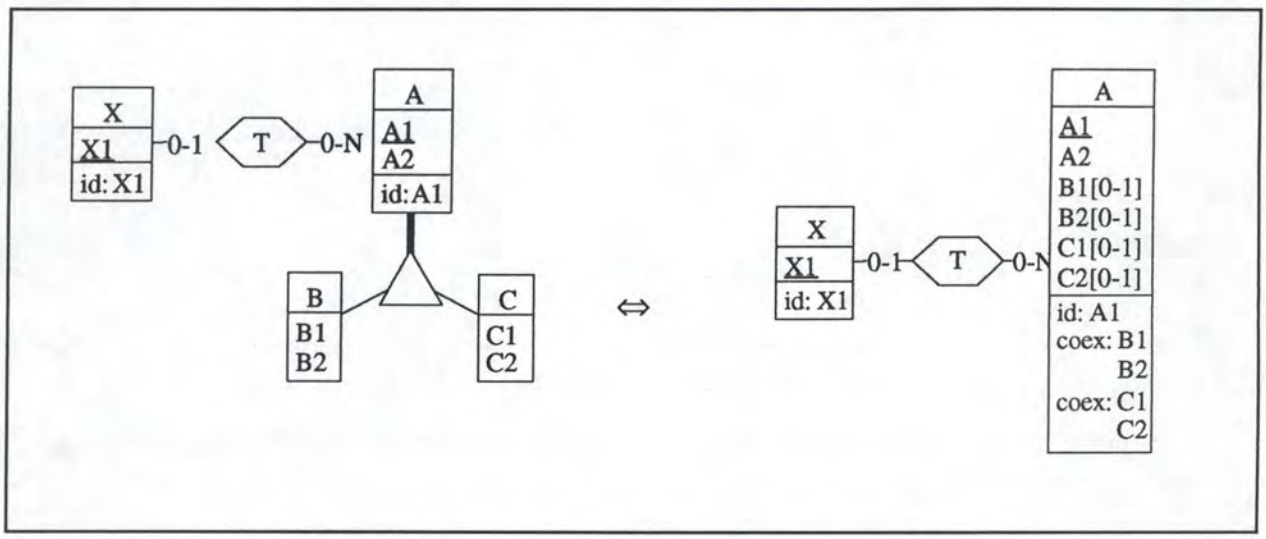


Figure 20 : Transformation par héritage ascendant

4.5 *Relation is-a où un sous-type au moins joue un rôle dans un T.A.*

La transformation d'une relation is-a où un sous-type au moins joue un rôle dans un T.A. peut se faire par héritage ascendant. Le rôle joué par le sous-type migre vers le surtype. Cette migration induit une contrainte d'implication qui indique que, quand il y a un T.A. T, il doit y avoir une valeur pour l'attribut B1.

4.5.1 *Relations intéressantes et leurs transformations*

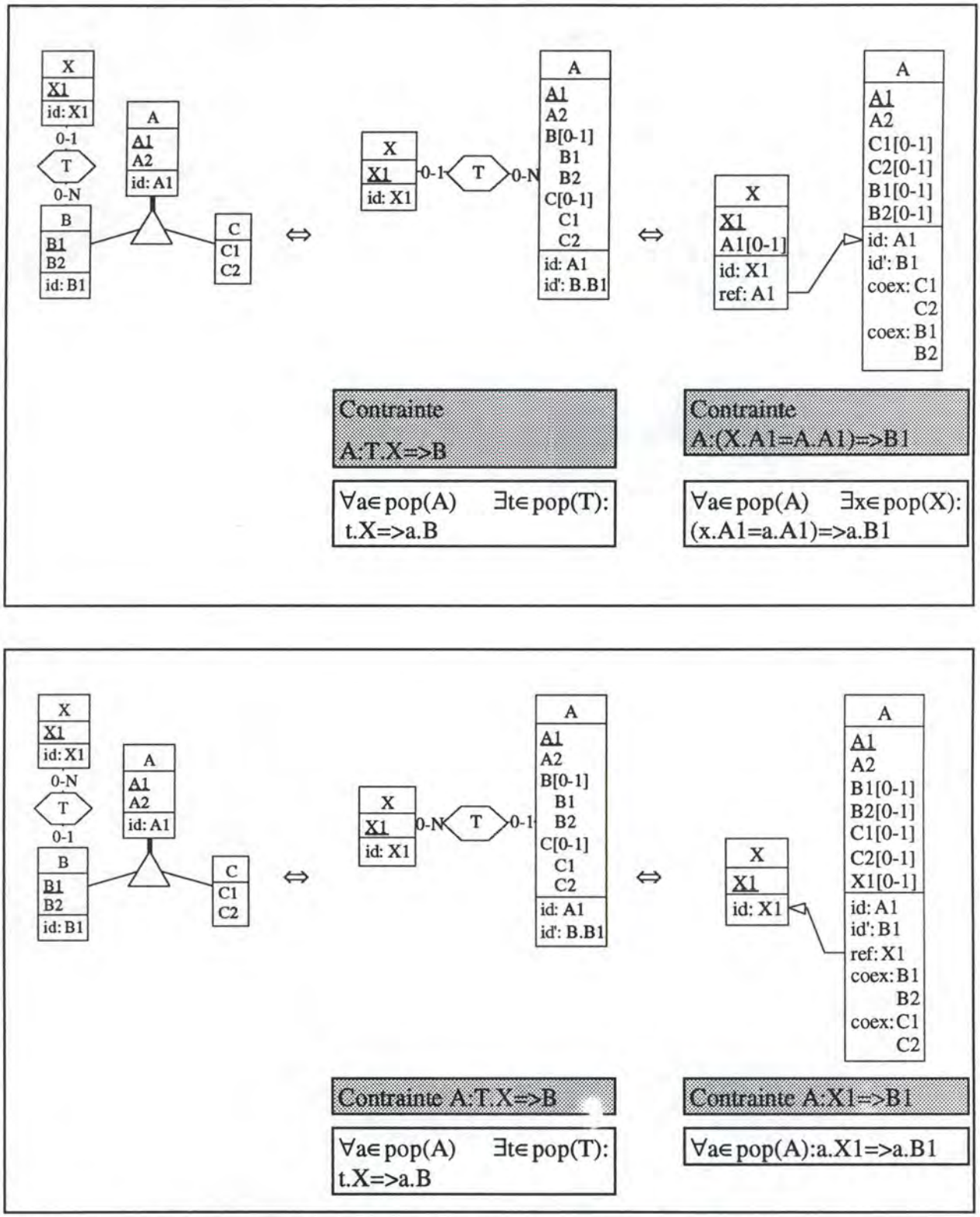


Figure 21 : Transformation par héritage ascendant de relation is-a (sous-type joue un rôle)

Si la cardinalité minimale du rôle joué par le T.E. X dans le T.A. T est 1, cela implique que la clé étrangère référençant le T.E. devient obligatoire; la cardinalité de l'attribut possède alors une cardinalité de [1-1].

Si la cardinalité minimale du rôle joué par le T.E. surtype dans le T.A. T est 1, cela implique une clé étrangère de type equal c'est-à-dire une clé étrangère avec une contrainte d'inclusion.

4.5.2 Code SQL

Les contraintes se traduisent en SQL par :

Pour le premier schéma :

```
alter table A add constraint IMP_A
check(not exists
      (select * from X
       where X.A1 = A.A1)
       or (B1 is not null)) ;
```

Pour le deuxième schéma :

```
alter table A add constraint IMP_A
check((X1 is null)
    or (B1 is not null))
```

5. *Etude des relations is-a transformées par la technique d'héritage descendant*[7],[8],[10]

5.1 *Relation is-a sans identifiant dans le surtype, ni dans les sous-types.*

Dans le cas de relation is-a sans identifiant dans le surtype ni dans les sous-types, la transformation de la relation is-a par héritage descendant n'est valide que pour les sous-types disjoints. Les autres cas ne permettent pas d'identifier les T.E. qui appartiennent aux sous-types.

S'il n'y a pas de contrainte de disjonction, alors le surtype doit avoir un identifiant pour que la relation is-a puisse être transformée (ce cas sera étudié dans le paragraphe suivant).

La contrainte de totalité implique que seuls les types d'entités provenant des sous-types restent après la transformation.

Les schémas obtenus après transformation sont conformes au modèle relationnel.

5.1.1 *Relations intéressantes et leurs transformations*

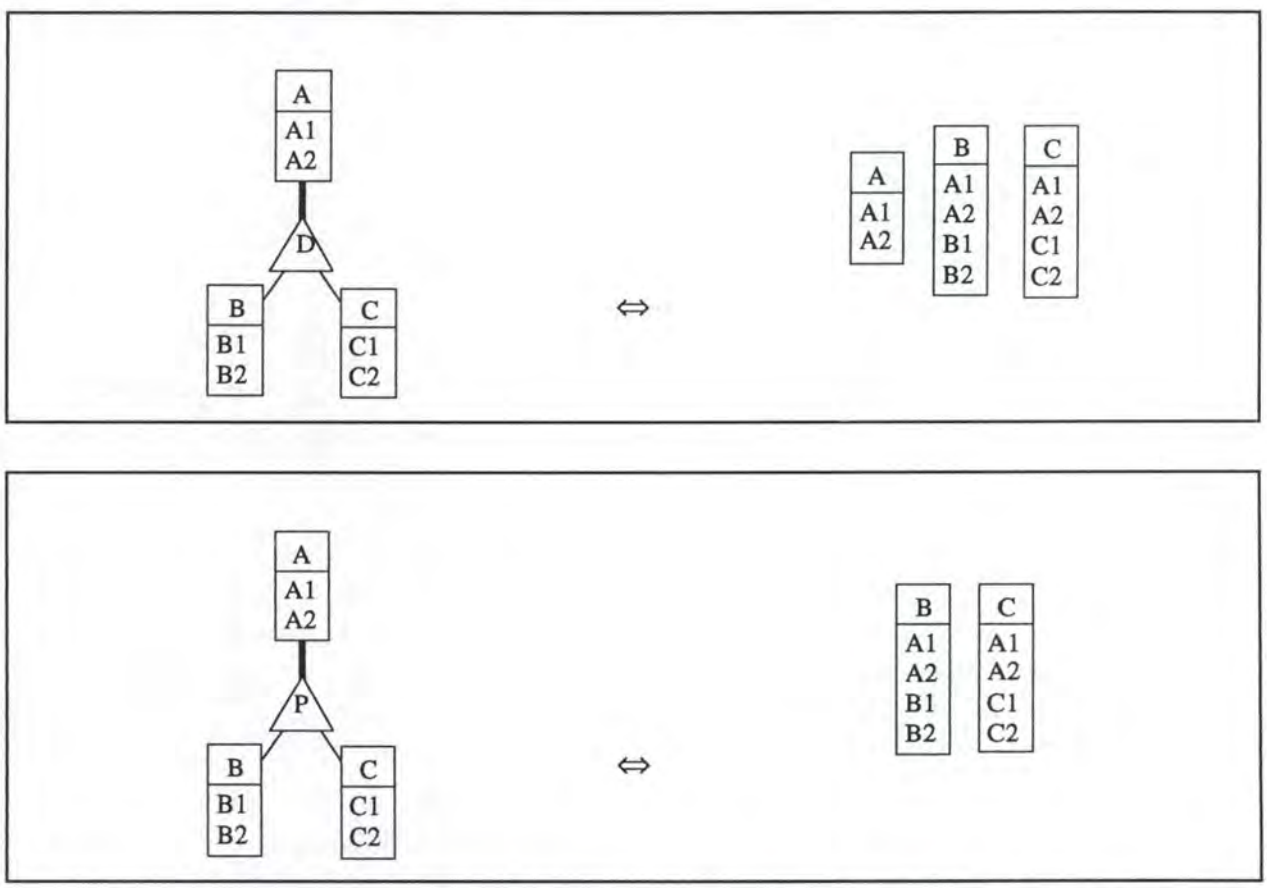


Figure 22 : Transformation par héritage descendant (pas d'identifiant dans surtype ni dans sous-types)

5.1.2 *Code SQL*

Le code pour le schéma avec la contrainte de disjonction est :

```
create table B (
   A1 char(1) not null,
   A2 char(1) not null,
   B1 char(1) not null,
   B2 char(1) not null);

create table A (
   A1 char(1) not null,
   A2 char(1) not null);

create table C (
   A1 char(1) not null,
   A2 char(1) not null,
   C1 char(1) not null,
   C2 char(1) not null);
```

Le code pour le schéma avec la contrainte de partition est :

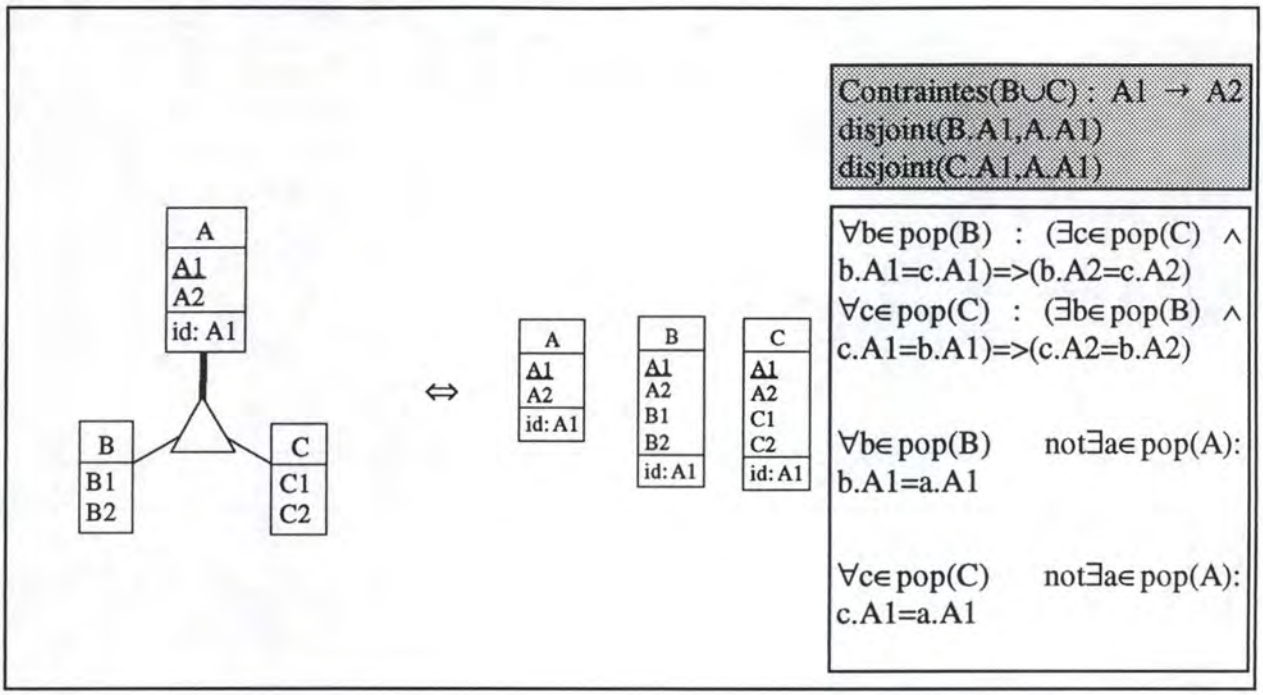
```
create table B (
   A1 char(1) not null,
   A2 char(1) not null,
   B1 char(1) not null,
   B2 char(1) not null);

create table C (
   A1 char(1) not null,
   A2 char(1) not null,
   C1 char(1) not null,
   C2 char(1) not null);
```

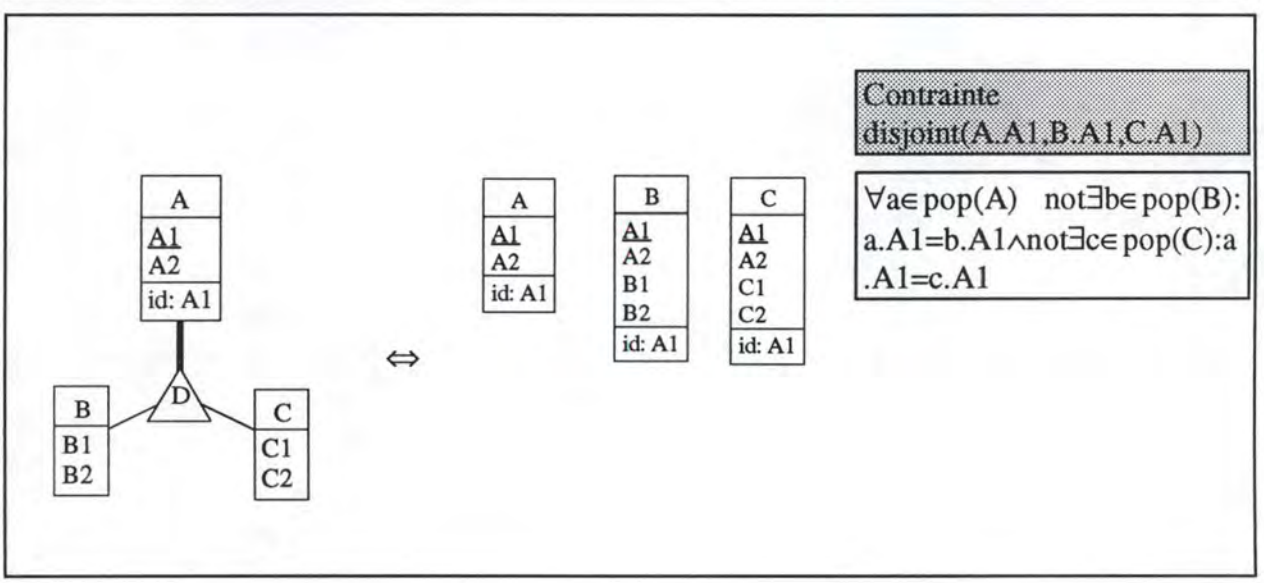
5.2 ***Relation is-a ayant un identifiant dans le surtype uniquement***

La transformation de relations is-a ayant un identifiant dans le surtype nous donne des schémas avec des contraintes qu'il faut traduire :

- Lorsque le surtype d'une relation is-a possède une contrainte de disjonction, une contrainte vient exprimer la disjonction entre les sous-types et surtype.
- Lorsque le surtype d'une relation is-a possède une contrainte de totalité, il doit y avoir une contrainte d'implication exprimant le fait que si des T.E. ont même valeur pour l'identifiant provenant du surtype, alors ils ont également mêmes valeurs pour les autres attributs provenant du surtype.
- Lorsque le surtype d'une relation is-a ne possède pas de contrainte, le schéma doit comporter des contraintes de disjonction et la contrainte d'implication citée plus haut
- Lorsque le surtype d'une relation is-a possède une contrainte de partition, seule une contrainte de disjonction entre les sous-types est présente.

5.2.1 *Relations intéressantes et leurs transformations*





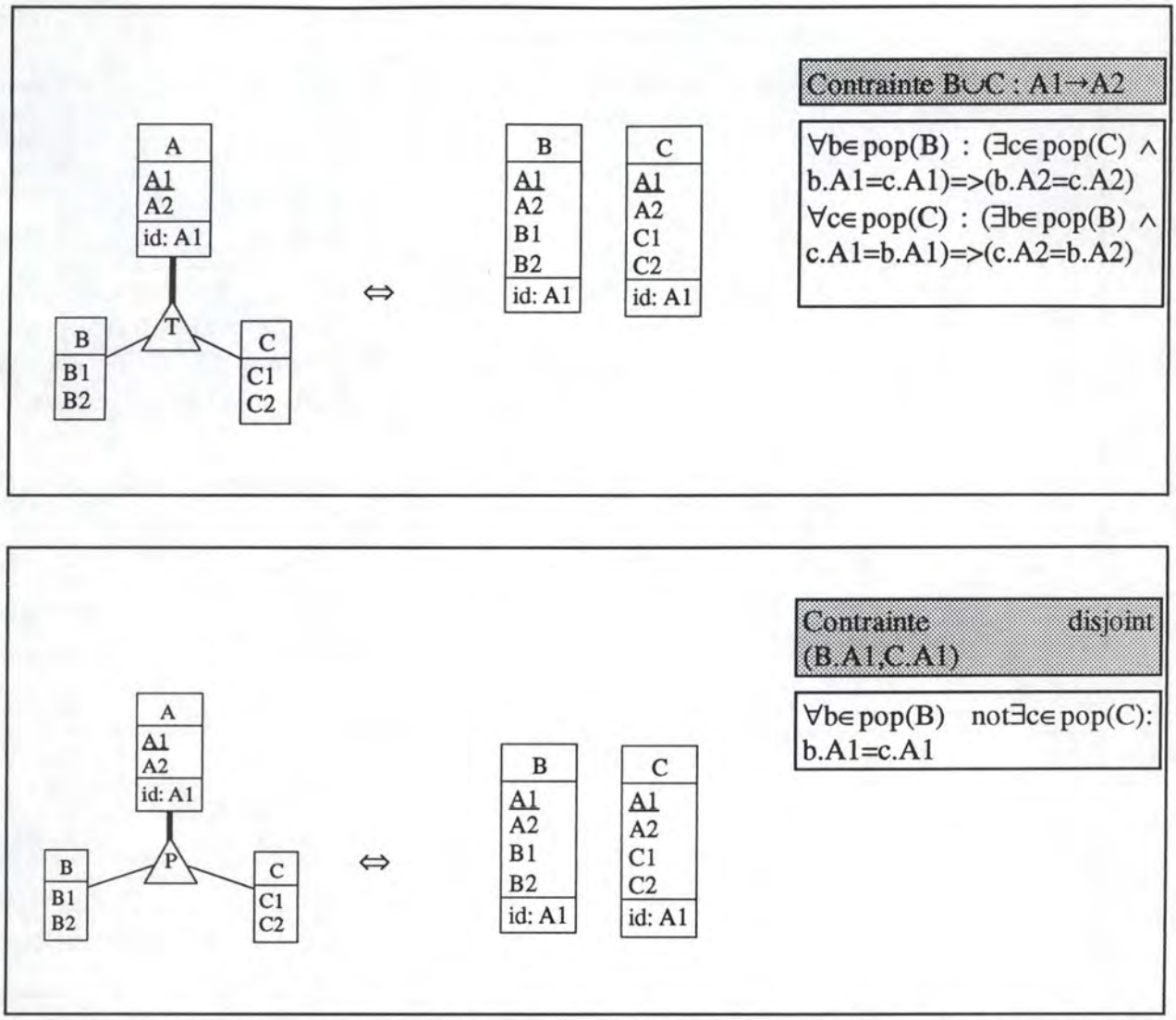


Figure 23: Transformation par héritage descendant de relation is-a (identifiant dans le surtype)

5.2.2 *Code SQL*

Pour la contrainte (B∪C∪…∪Z) : A1→A2, il faut rajouter le code suivant :

```
alter table B add constraint IMP_B
check((not exists
        (select * from C
        where C.A1 = B.A1 and C.A2 <> B.A2)
        and…
        and (not exists
        (select * from Z
        where Z.A1=B.A1 and Z.A2<>B.A2))
```

```
alter table C add constraint IMP_C
check((not exists
        (select * from B
         where B.A1 = C.A1 and B.A2 <> C.A2))
         and...
         and (not exists
         (select * from Z
          where Z.A1=C.A1 and Z.A2<>C.A2))

...

alter table Z add constraint IMP_Z
check((not exists
        (select * from B
         where B.A1 = Z.A1 and B.A2 <> Z.A2))
         and (not exists
         (select * from C
          where C.A1=Z.A1 and C.A2<>Z.A2))
         and...
```

Pour la contrainte disjoint(A.A1,B.A1,...,Z.A1), il faut rajouter le code suivant :

```
alter table A add constraint DIS_A_B_..._Z
check(A1 not in
      (select A1 from B)
      and A1 not in
      (select A1 from C)
      and...
      and A1 not in
      (select A1 from Z)) ;

alter table B add constraint DIS_B_C_...Z_A
check(A1 not in
       (select A1 from A)
        and (A1 not in
        (select A1 from C)
        and...
        and (A1 not in
        (select A1 from Z)) ;

...

alter table Z add constraint DIS Z_A_B_C_...
check(A1 not in
       (select A1 from A)
        and (A1 not in
        (select A1 from B)
         and (A1 not in
        (select A1 from C))
        and... ;
```

5.3 *Relation is-a n'ayant pas d'identifiant dans tous les sous-types et n'ayant pas d'identifiant dans le surtype*

Comme vu précédemment, pour pouvoir faire les transformations de relations is-a ayant une contrainte de disjonction, il faut que le surtype ait un identifiant.

Cette étude est donc identique à celle du paragraphe 5.1

5.4 *Relation is-a où le surtype joue un rôle dans un T.A.*

Quand la technique d'héritage descendant est utilisée pour transformer une relation is-a dont le surtype joue un rôle dans un T.A., ce rôle est remplacé par un rôle multi-T.E. (R.bac). Ce rôle est tenu par le surtype et les sous-types s'il y a recouvrement des sous-types (c'est-à-dire s'il n'y a pas de contrainte de totalité), sinon par les sous-types uniquement. Les contraintes qui découlent de ces transformations sont identiques à celles reprises dans le paragraphe 5.2.

Le schéma est rendu relationnel en scindant le T.A. R comme présenté au paragraphe 1.5. Le résultat est un schéma possédant toujours des structures non conformes au schéma relationnel, des types d'associations one-to-many, qu'il faut traduire en clés étrangères.

La Figure 24 présente les résultats de la transformation par héritage descendant d'une telle relation is-a. Cette figure ne reprend que les relations is-a où une instance du surtype participe 0 à N fois au T.A. Pour les relations is-a où une instance du surtype participe 0 à 1 fois ou 1 et 1 fois au T.A., l'analyse est semblable.

5.4.1 *Relations intéressantes et leurs transformations*

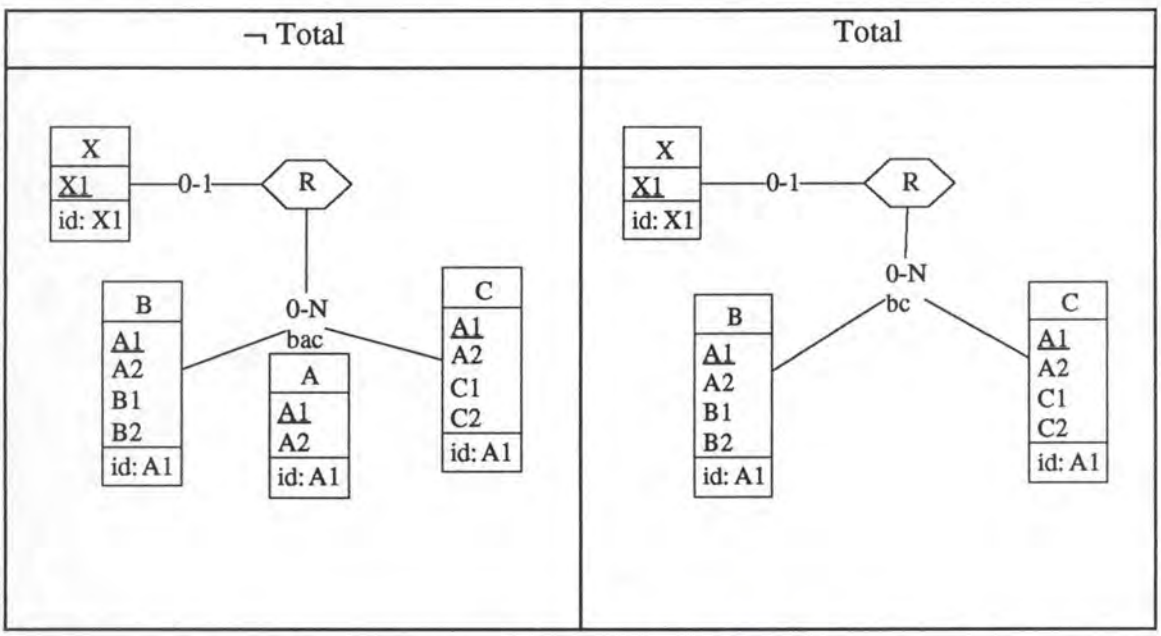


Figure 24: Transformation par la technique d'héritage descendant (contraintes de la Figure 23 non reprises).

5.4.2 *Code SQL*

Le code pour les contraintes est identique à celui présenté au paragraphe 5.2.

5.5 *Relations is-a où un des sous-types au moins joue un rôle dans un T.A.*

La technique de transformation par héritage descendant préserve les sous-types aussi bien que leurs rôles. Cette transformation ne pose donc aucun problème avec les relations is-a dont un sous-type joue un rôle dans un T.A..

6. Résumé et conclusion

6.1 Résumé

Grâce à l'étude précédente, nous pouvons tirer les conclusions suivantes :

- Pour les relations is-a ne possédant pas d'identifiant ni dans le surtype, ni dans les sous-types, la transformation de la relation is-a ne peut se faire que par héritage ascendant.

  Si l'on veut utiliser des méthodes de transformation donnant des clés étrangères, il faut prévoir l'introduction d'un identifiant technique dans le surtype.

  La transformation par héritage descendant ne peut se faire que pour les relations is-a dont les sous-types sont disjoints. Pour permettre la transformation par une telle méthode, de relations is-a dont le surtype n'est pas soumis à une contrainte ou soumis à une contrainte de totalité, il est nécessaire d'introduire un identifiant technique dans le surtype.

- Pour les relations is-a ayant un identifiant dans le surtype, la transformation par matérialisation peut se faire avec création de clés étrangères. Des contraintes viennent compléter les transformations. La transformation introduisant un attribut TYPE dans le surtype a l'avantage de simplifier les checks à générer, même s'il en découle l'introduction d'un trigger.

  Dans le cas de l'héritage ascendant, la transformation peut se faire.

  La transformation par héritage descendant est possible mais génère des contraintes.

- Lorsque le surtype joue un rôle dans une relation, les transformations par matérialisation et par héritage ascendant ne posent pas de problème. Si l'on veut permettre la création de clés étrangères lors de cette transformation, un identifiant est nécessaire dans le surtype et il faut ajouter un identifiant technique si nécessaire.

  Pour la transformation par héritage descendant, un T.A. ayant un rôle multi-T.E. est créé ainsi que des contraintes. On transforme ensuite ce T.A. en la scindant.

- Les relations is-a dont un sous-type joue un rôle dans un T.E. peuvent être transformées par matérialisation en créant des clés étrangères, si au moins un des T.E. jouant un rôle dans le T.A. a un identifiant.

  Si l'on veut faire cette transformation par héritage ascendant, le rôle joué par le sous-type dans l'association doit, après transformation, être joué par le surtype. Une contrainte en découle.

  La transformation par héritage descendant ne pose pas de problème.

Le Tableau 2 reprend les transformations étudiées en colonnes, les relations is-a en lignes et à l'intersection, la possibilité ou non de la transformation de ce type de relations is-a.

	Matérialisation	**héritage ascendant**	**héritage descendant**
Sans identifiant dans le surtype	non	oui	oui (si les sous-types sont non disjoints)
Identifiant dans le surtype	oui	oui	oui
Pas d'identifiant dans tous les sous-types ni dans le surtype	non	oui	oui (si les sous-types sont non disjoints)
Surtype joue un rôle	non	oui	oui
Sous-type joue un rôle	oui (si identifiant dans un des T.E. jouant un rôle)	oui	oui

Tableau 2 : Transformations possibles

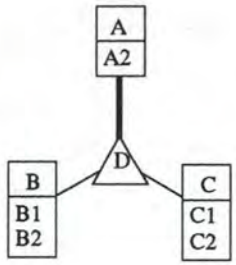
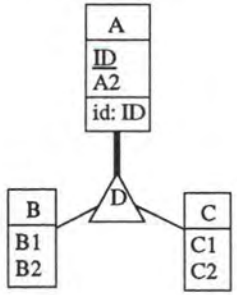
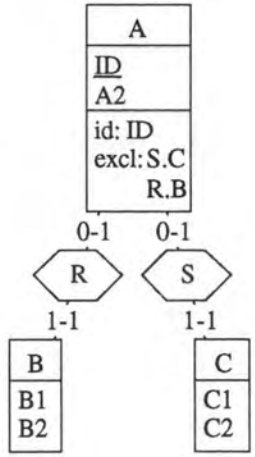
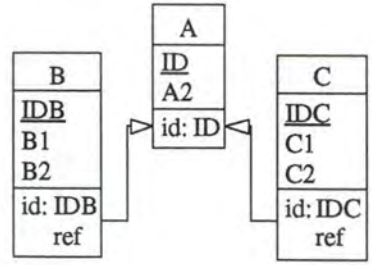
Dans le Tableau 3, l'intersection d'une ligne et d'une colonne donne les éléments nécessaires dans la relation is-a pour que la transformation puisse être effectuée correctement et complètement.

	Matérialisation		héritage ascendant	héritage descendant
	simple ou avec création d'att. booléens	*création attribut TYPE*		
Sans identifiant dans le surtype	id. technique dans le surtype	id. technique dans le surtype		id. technique dans le surtype ou sous-types non disjoints
Identifiant dans le surtype				
Pas d'identifiant dans le surtype tous les sous-types n'ont pas d'identifiant	id. technique dans le surtype (F.K. directe) ou dans tous les sous-types (F.K. inverse)	id. technique dans le surtype		id. technique dans le surtype ou sous-types non disjoints
Surtype joue un rôle	identifiant dans le surtype	identifiant dans le surtype		id. dans le surtype ou sous-types non disjoints
Sous-type joue un rôle	identifiant dans un des T.E. jouant un rôle	identifiant dans un des T.E. jouant un rôle		id. dans le surtype ou sous-types non disjoints

Tableau 3 : Eléments nécessaires aux transformations

En résumé, voici les différents cas généraux, leurs transformations et le code SQL qui en découle. Nous n'avons repris que les cas de relations is-a où le surtype possède une contrainte de disjonction. Pour les autres contraintes, les transformations et le code SQL peuvent être facilement déterminés.

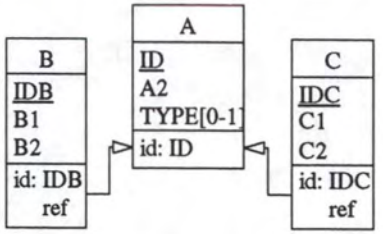
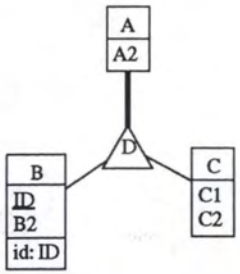
De plus, le code SQL repris dans ces tableaux ne concerne que les contraintes.

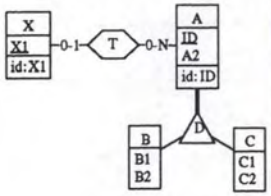
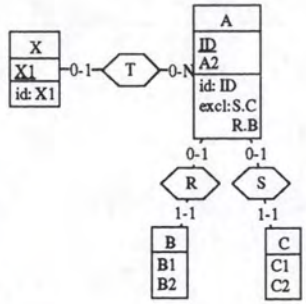
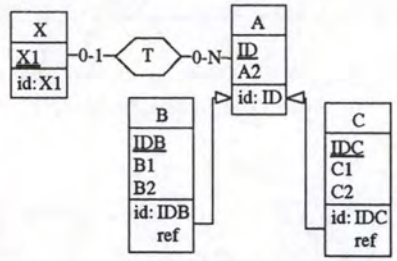
<table>
<tr><th colspan="4">Transformation de matérialisation</th></tr>
<tr><th colspan="2">Relation is-a</th><th>Première transformation</th><th>Deuxième transformation</th><th>code SQL</th></tr>
<tr><th>Type</th><th>Représentation</th><th></th><th></th><th></th></tr>
<tr><td>pas d'id.
dans le surtype ni dans les sous-types</td><td> 
</td><td>Impossible</td><td></td><td></td></tr>
<tr><td>id. dans le surtype</td><td> 
</td><td> 
</td><td>(création f.k.)
 

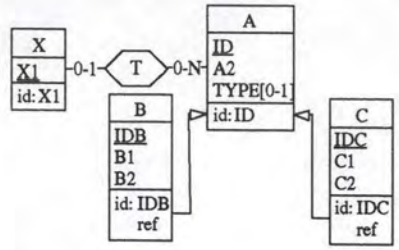
Contrainte disjoint(B.IDB,C.IDC)[3]</td><td>alter table B add constraint DIS_B
check(IDB not in
(select IDC from C)) ;

alter table C add constraint DIS_C
check(IDC not in
(select IDB from B)) ;</td></tr>
</table>

[3] IDB, IDC ont même valeur que ID de A. Nous avons voulu prévoir les cas où, lors de la création de ces attributs dans B et C, il y ait déjà un attribut avec les mêmes noms.

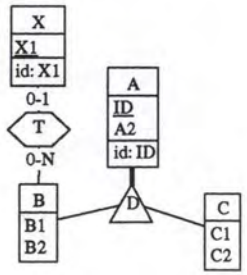
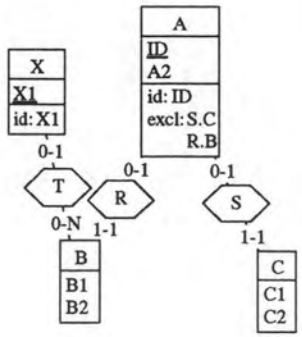
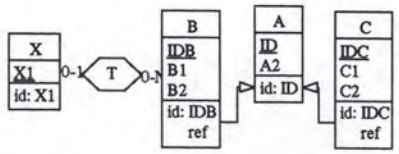
			(création att. TYPE)  	alter table B add constraint TYPE_B check (not exists (select TYPE from A where A.ID = B.IDB and TYPE not like '%"B"%')) ; alter table C add constraint TYPE_C check (not exists (select TYPE from A where A.ID = C.IDC and TYPE not like '%"C"%')) ;
pas d'id. dans le surtype, pas d'id. dans tous les sous-types.	 	Impossible		

le surtype joue un rôle			*(création clés étrangères)*  Contrainte disjoint(B.IDB,C.IDC) le T.A. T doit être transformé en clés étrangères pour que le schéma soit relationnel	alter table B add constraint DIS_B check(IDB not in (select IDC from C)) ; alter table C add constraint DIS_C check(IDC not in (select IDB from B)) ;

<table>
<tr>
<td></td>
<td></td>
<td></td>
<td>(création att. TYPE)
 

Contrainte Dom(TYPE)={'B'}∪{'C'}

le T.A. T doit être transformé en clés étrangères pour que le schéma soit relationnel</td>
<td>alter table B add constraint TYPE_B
check(not exists
(select TYPE from A where A.ID=B.IDB
and TYPE not like '%"B"%')) ;

alter table C add constraint TYPE_C
check(not exists
(select TYPE from A where A.ID=C.IDC
and TYPE not like '%"C"%'))</td>
</tr>
<tr>
<td>un sous-type joue un rôle</td>
<td> 
</td>
<td> 
</td>
<td>(création clés étrangères)
 

Contrainte disjoint(B.A1,C.A1)

le T.A. T doit être transformé en clés étrangères pour que le schéma soit relationnel</td>
<td>alter table B add constraint DIS_B
check(IDB not in
(select IDC from C)) ;

alter table C add constraint DIS_C
check(IDC not in
(select IDB from B))</td>
</tr>
</table>

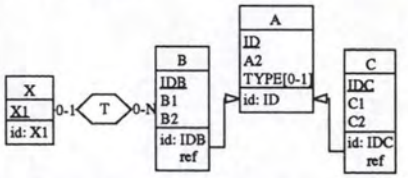
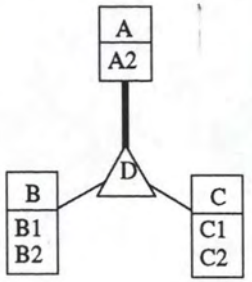
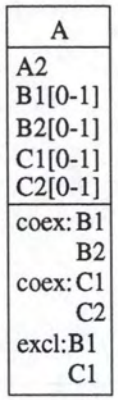
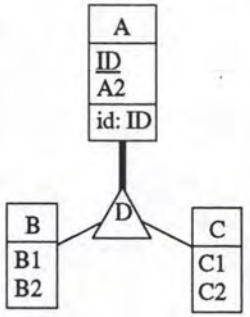
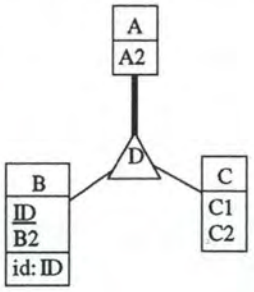
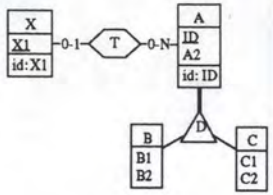
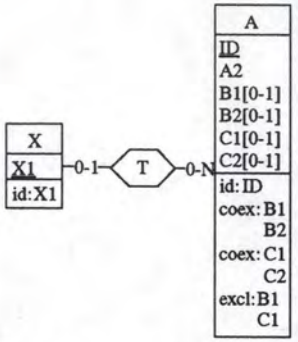
			(*création att. TYPE*)  Contrainte Dom(TYPE)={'B'}∪ {'C'} le T.A. T doit être transformé en clés étrangères pour que le schéma soit relationnel	alter table B add constraint TYPE_B check(not exists (select TYPE from A where A.ID=B.IDB and TYPE not like '%"B"%')) ; alter table C add constraint TYPE_C check(not exists (select TYPE from A where A.ID=C.IDC and TYPE not like '%"C"%')) ;

Tableau 4 : Transformation de matérialisation de relation is-a

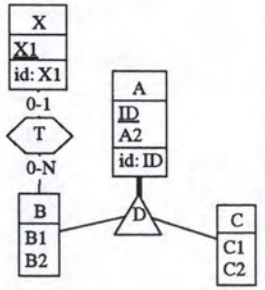
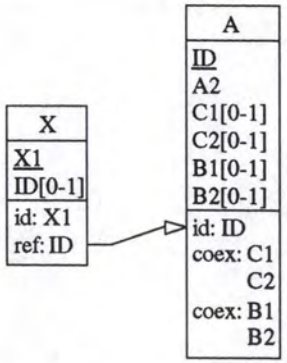
Transformation par héritage ascendant			
relation is-a		***transformation***	***code SQL***
Type	***représentation***		
pas d'id. dans le surtype ni dans les sous-types			alter table A add constraint COEXA check((B1 is not null and B2 is not null) or (B1 is null and B2 is null)); alter table A add constraint COEXA_1 check((C1 is not null and C2 is not null) or (C1 is null and C2 is null)); alter table A add constraint ISAA check((C1 is not null and B1 is null) or (C1 is null and B1 is not null) or (C1 is null and B1 is null));

id. dans le surtype		A ID A2 B1[0-1] B2[0-1] C1[0-1] C2[0-1] id: ID coex: B1 B2 coex: C1 C2 excl: B1 C1	alter table A add constraint COEXA check((B1 is not null and B2 is not null) or (B1 is null and B2 is null)); alter table A add constraint COEXA_1 check((C1 is not null and C2 is not null) or (C1 is null and C2 is null)); alter table A add constraint ISAA check((C1 is not null and B1 is null) or (C1 is null and B1 is not null) or (C1 is null and B1 is null));
pas d'id. dans le surtype pas d' id. dans tous les sous-types		A A2 ID[0-1] B2[0-1] C1[0-1] C2[0-1] id': ID coex: ID B2 coex: C1 C2 excl: ID C1	alter table A add constraint COEXA check((ID is not null and B2 is not null) or (ID is null and B2 is null)); alter table A add constraint COEXA_1 check((C1 is not null and C2 is not null) or (C1 is null and C2 is null)); alter table A add constraint ISAA check((C1 is not null and ID is null) or (C1 is null and ID is not null) or (C1 is null and ID is null));

<table>
<tr>
<td>le surtype joue un rôle</td>
<td> 
</td>
<td> 

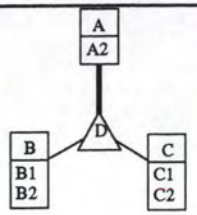
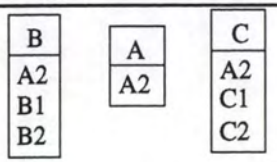
le T.A. T doit être transformé en clés étangères pour que le schéma soit relationnel</td>
<td>alter table A add constraint COEXA
check((B1 is not null and B2 is not null)
or (B1 is null and B2 is null));

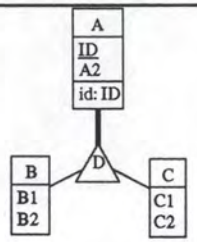
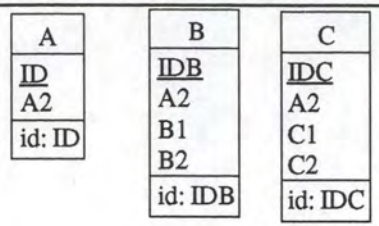
alter table A add constraint COEXA_1
check((C1 is not null and C2 is not null)
or (C1 is null and C2 is null));

alter table A add constraint ISAA
check((C1 is not null and B1 is null)
or (C1 is null and B1 is not null)
or (C1 is null and B1 is null));</td>
</tr>
<tr>
<td>un sous-type joue un rôle</td>
<td> 
</td>
<td> 

Contrainte
A:(X.ID=A.ID)=>B1</td>
<td>alter table A add constraint IMP_A
check(not exists
(select * from X
where X.ID = A.ID)
or (B1 is not null)) ;</td>
</tr>
</table>

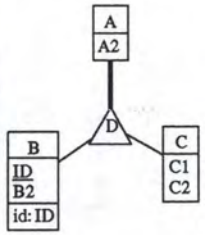
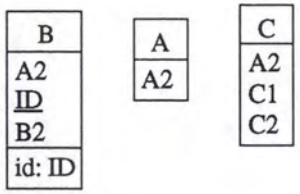
Tableau 5 : Transformation par héritage ascendant de relation is-a

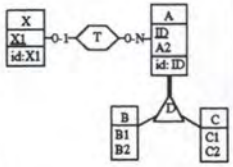
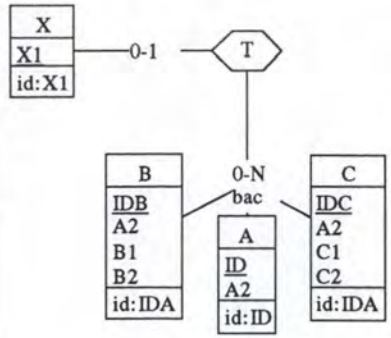
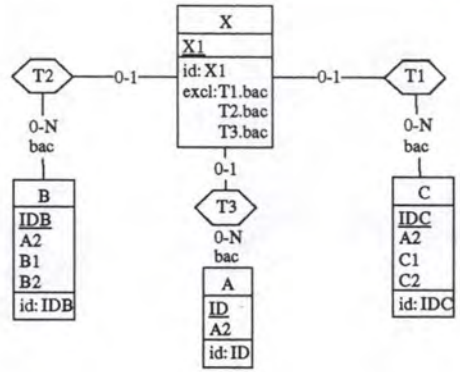
Transformation par héritage descendant				
relation is-a		*Première transformation*	*Deuxième transformation*	*Code SQL*
Type	*Représentation*			
pas d'id. dans le surtype ni dans les sous-types				

<table>
<tr>
<td>id. dans le surtype</td>
<td> 
</td>
<td> 

Contrainte
disjoint(A.A1,B.IDB,C.IDC)</td>
<td></td>
<td>alter table A add constraint DIS_A_B_C
check(ID not in
(select IDA from B)
and ID not in
(select IDA from C)) ;

alter table B add constraint DIS_B_C_A
check(IDB not in
(select ID from A)
and IDB not in
(select IDC from C)) ;

alter table C add constraint DIS_C_B_A
check(IDC not in
(select IDB from B)
and IDC not in
(select ID from A))</td>
</tr>
<tr>
<td>pas d'id. dans le surtype
pas d'id. dans tous les sous-types</td>
<td> 
</td>
<td> 
</td>
<td></td>
<td></td>
</tr>
</table>

un sous-type joue un rôle		 Disjoint(A.ID,B.IDB,C.IDC)	 Disjoint(A.ID,B.IDA,C.IDA) le T.A. T doit être transformé en clés étrangères pour que le schéma soit relationnel	alter table A add constraint DIS_A_B_C check(ID not in (select IDB from B) and ID not in (select IDC from C)) ; alter table B add constraint DIS_B_C_A check(IDB not in (select ID from A) and IDB not in (select IDC from C)) ; alter table C add constraint DIS_C_B_A check(IDC not in (select IDB from B) and IDC not in (select ID from A)) ;

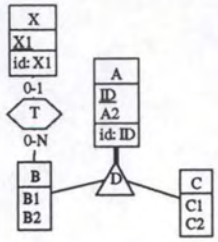
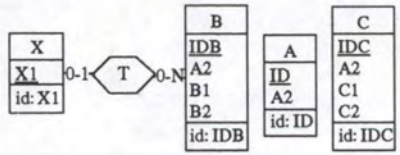
le surtype joue un rôle		 le T.A. T doit être transformé en clés étrangères pour que le schéma soit relationnel		alter table A add constraint DIS_A_B_C check(ID not in (select IDB from B) and ID not in (select IDC from C)) ; alter table B add constraint DIS_B_C_A check(IDB not in (select ID from A) and IDB not in (select IDC from C)) ; alter table C add constraint DIS_C_B_A check(IDC not in (select IDB from B) and IDC not in (select ID from A)) ;

Tableau 6 : Transformation par héritage descendant de relation is-a

6.2 *Algorithme de transformation*

Pour automatiser la transformation de relation is-a, nous proposons d'illustrer la suite des actions à effectuer par un algorithme (Figure 25).

L'utilisateur peut choisir une transformation : soit la transformation de matérialisation (MAT), soit la transformation par héritage ascendant (ASC), soit la transformation par héritage descendant (DESC).

Si la transformation demandée est la matérialisation, il faut encore choisir parmi la matérialisation simple (où les contraintes éventuelles sont exprimées sur les sous-types directement) (FK), par création d'attributs booléens dans le surtype (BOOL) ou par la technique de l'indicateur de sous-type (INDIC).

Les variables, les fonctions et les procédures utilisées dans cet algorithme sont présentés dans les Tableau 7, Tableau 8 et Tableau 9

Nom de la variable	**Contenu de la variable**	**Domaine de la variable**
transfo-demandée	la transfo. désirée par l'utilisateur	MAT : transfo. par matérialisation ASC : transfo. par héritage ascendant DESC : transfo. par héritage descendant.
type_transfo_mat	le type de transfo. par matérialisation demandée	FK : transfo. avec contraintes exprimées directement sur les sous-types. BOOL : transfo. avec créations d'attributs booléens INDIC : transfo. par la méthode de l'indicateur de sous-types
surtype	le nom du surtype	

Tableau 7 : Variables utilisées dans l'algorithme

Fonction	Arguments	Résultats
SansIdentifiant(type_entités)	le nom d'un T.E.	un booléen indiquant s'il y a un identifiant dans le T.E. ou non
JoueRôle(type_entités)	le nom d'un T.E.	un booléen indiquant si le T.E. joue un rôle ou non dans un T.A.
SstypesDisjoints	le nom d'un surtype	un booléen indiquant si les sous-types sont disjoints ou non

Tableau 8 : Fonctions utilisées dans l'algorithme

Procédure	Résultats
AjoutFK	réalise la transfo. par matérialisation (où les contraintes exprimées sur les sous-types directement)
AjoutIndic	réalise la transfo. par matérialisation par la méthode de l'indicateur de sous-types
AjoutBool	réalise la transfo. par matérialisation par la création d'attributs booléens dans le surtype
MultiTE	réalise la transfo. d'un rôle en un rôle multi-TE
Split	réalise la transfo. d'un rôle multi-TE en scindant le T.A.
HérAsc	réalise la transfo. par héritage ascendant
HérDesc	réalise la transfo. par héritage descendant
AjoutIdTech(type_entités)	ajoute un identifiant technique au T.E. qu'elle prend en argument

Tableau 9 : Procédures utilisées dans l'algorithme

```
Case transfo_demandée of
{
MAT :   Case type_transfo_mat of
        {BOOL :     if SansIdentifiant(surtype) then
                        {
                         AjouterIdTech (surtype)
                        }
                    AjoutBool
        'FK ':      if SansIdentifiant(surtype) then
                        {
                         AjouterIdTech (surtype)
                        }
                    AjoutFK
        'INDIC ':   if SansIdentifiant(surtype) then
                        {
                        AjouterIdTech (surtype)
                        }
                    AjoutIndic
        }
ASC :   HérAsc
DESC :if JoueRôle(surtype) then
                {
                if SansIdentifiant(surtype) then
                        {
                        AjoutIdTech (surtype)
                        }
                MultiTE
                Split
                }
                else
                {
                if SstypesDisjoints (surtype) then
                        {
                        HérDesc
                        }
                        else
                        {
                        if SansIdentifiant(surtype) then
                                {
                                  AjouterIdTech(surtype)
                                }
                        HérDesc
                        }
                }
        }
}
```

Figure 25 : Algorithme de transformation de relations is-a

6.3 *Exemples*

Dans ce paragraphe, nous allons mettre en application les différentes techniques étudiées dans le chapitre précédent.

6.3.1 Exemple 1

La Figure 26 montre un exemple de relation is-a où les sous-types jouent un rôle dans des T.A. Cette représentation pourrait concerner par exemple la préparation d'un dossier reprenant les étudiants faisant un doctorat ou déjà licenciés ou ceux qui vont l'être.

Le surtype, le T.E. *Etudiant* a pour attribut *N°* qui est identifiant, et *Nom*. Ce surtype ne subit pas de contrainte particulière : un *Etudiant* peut faire un *Doctorat* ou être *Licencié* ou simple *Etudiant*. Le sous-type *Doctorat* comprend l'attribut *TitreThèse* et le sous-type *Licencié* comprend l'attribut *TitreMém*. Le T.E. *Doctorat* joue un rôle dans un T.A. *réalisation* qui porte également sur le T.E. *Faculté*. Ce T.E. possède un attribut : *Nom*.

Le T.E. *Licencié* joue un rôle dans un T.A. *promotion* qui porte également sur le T.E. *Promoteur* qui possède les attributs *Nom*, *Prénom* et *Dptmt* (département).

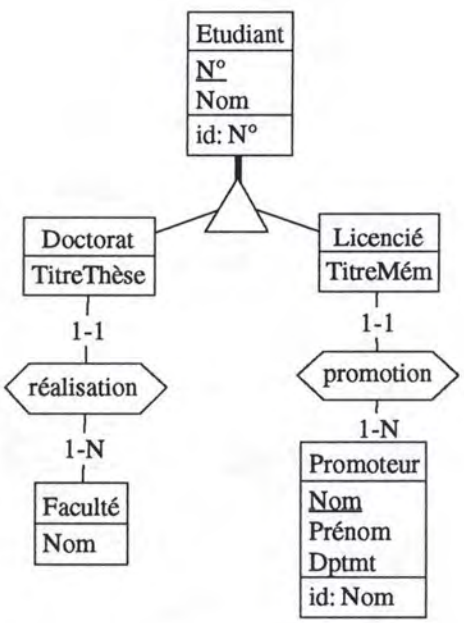


Figure 26 : Exemple de relation is-a (1)

Transformation par matérialisation

Cette relation is-a peut être transformée par la méthode de matérialisation simple. Comme le surtype possède un identifiant, nous pouvons appliquer la transformation sans modifier la relation is-a. Le résultat d'une telle transformation est donné à la Figure 27.

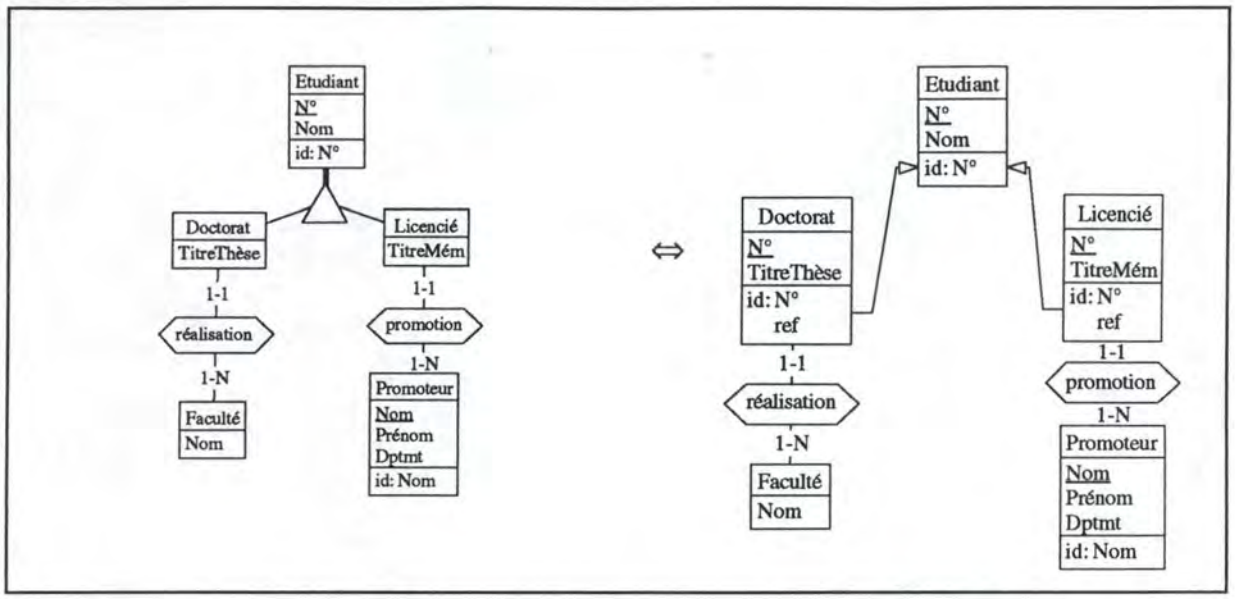


Figure 27 :Transformation par matérialisation simple

Il reste à transformer les T.A. one-to-many en clés étrangères pour avoir un schéma relationnel (Figure 28). Remarquons qu'une clé étrangère inverse est créée entre *Doctorat* et *Faculté* parce que seul le T.E. *Doctorat* possède un identifiant.

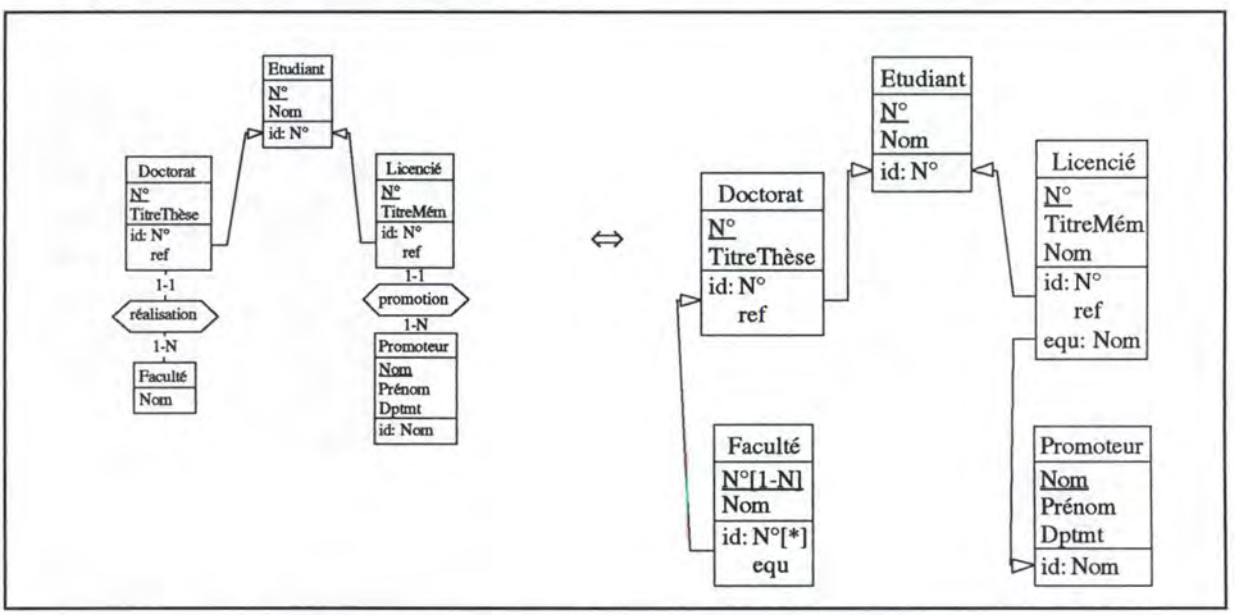


Figure 28 : Transformation des T.A. en clés étrangères

Si l'on veut utiliser la méthode de l'indicateur de sous-types, un attribut *TYPE* est créé dans le surtype et sa cardinalité est [0-N](Figure 29).

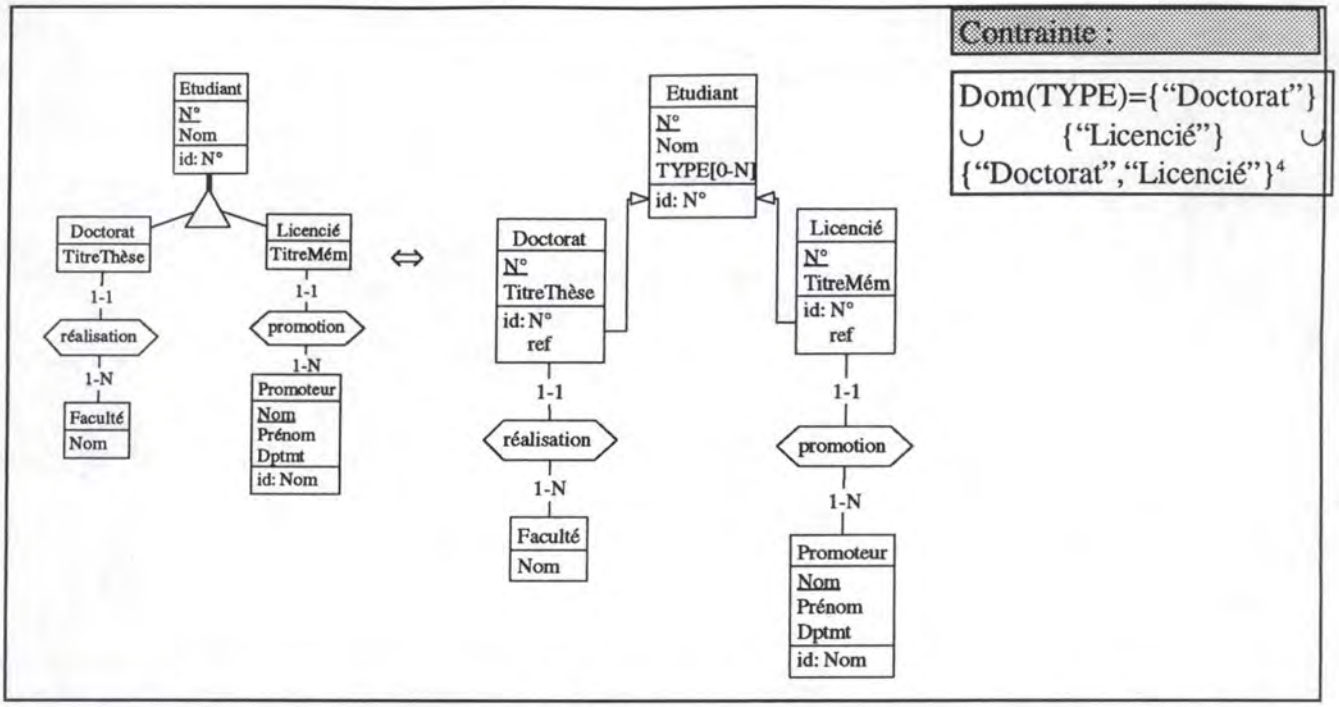
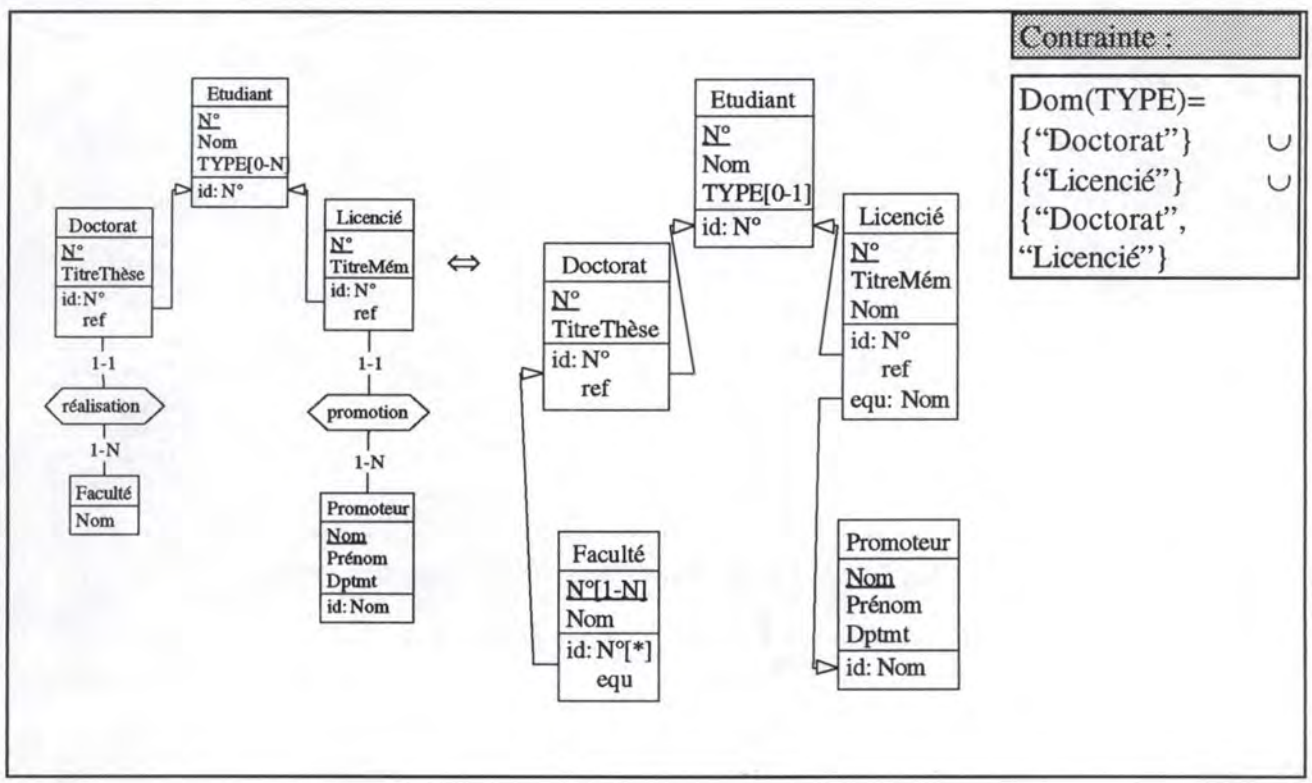


Figure 29 : Transformation par matérialisation par la méthode de l'indicateur de sous-types.

Pour rendre ce schéma relationnel, il est nécessaire de transformer l'attribut *TYPE* multivalué en un attribut monovalué *TYPE* par la méthode de concaténation et de transformer les T.A. en clés étrangères.



[4]Le code correspondant à cette contrainte peut être déterminé à partir du paragraphe 3.2

Figure 30 : Transformation de l'attribut multivalué et des T.A. en clés étrangères

Transformation par héritage ascendant

La méthode de transformation par héritage ascendant donne les résultats présentés à la Figure 31.

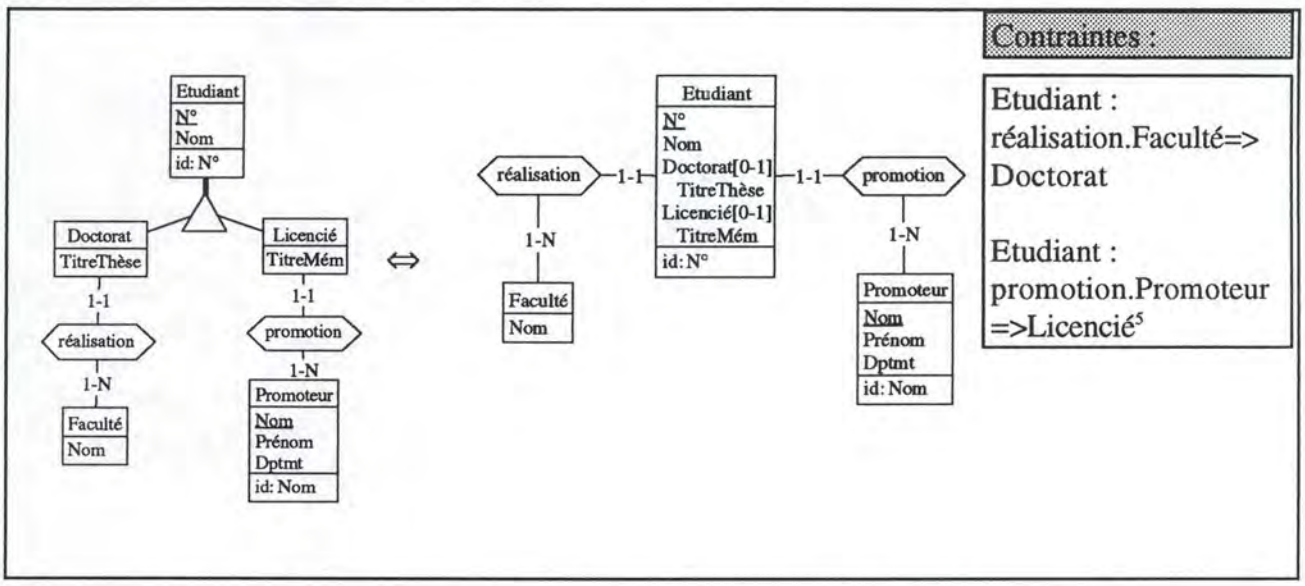


Figure 31 : Transformation par matérialisation par héritage ascendant

Le schéma relationnel correspondant à cette figure peut être trouvé en Figure 32 : les T.A. ont été transformés en clés étrangères et les attributs *Doctorat* et *Licencié* doivent être décomposés.

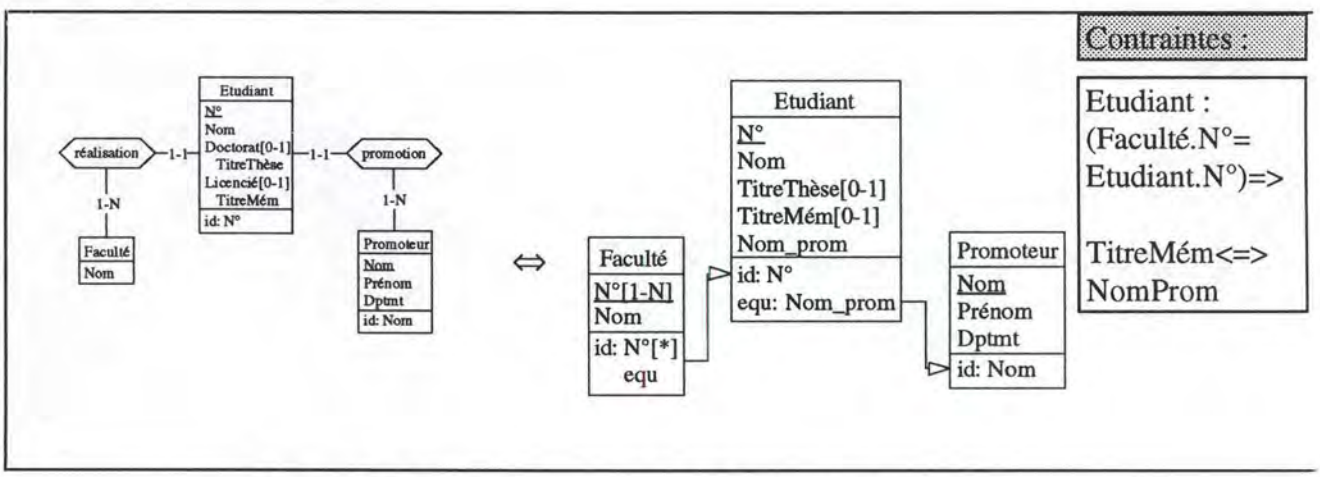


Figure 32: Transformation des T.A. en clés étrangères

[5] Le code correspondant à ces contraintes peut être déterminé à partir du paragraphe 4.5

Transformation par héritage descendant

La méthode de transformation par héritage descendant est la plus complexe. Le surtype joue un rôle et il possède un identifiant. Il est nécessaire de passer par une transformation des rôles des T.A. en rôles multi-T.E. puis de scinder la T.A formée.

Cette transformation donne des contraintes de disjonction entre sous-types et surtype et une contrainte d'implication. Le résultat de cette transformation est présenté à la Figure 33.

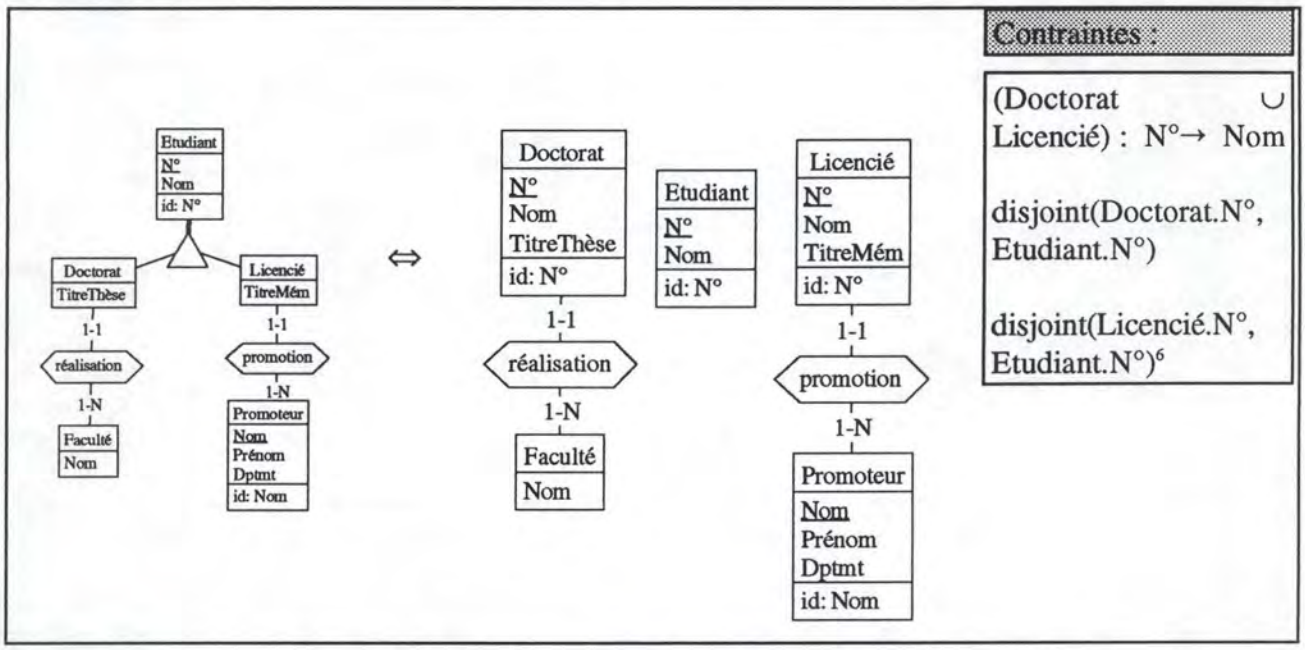


Figure 33 : Transformation par héritage descendant

Ce schéma doit encore être transformé pour être relationnel. La Figure 34 présente les transformations des T.A.

[6] Le code correspondant à ces contraintes peut être déterminé à partir du paragraphe 5.2

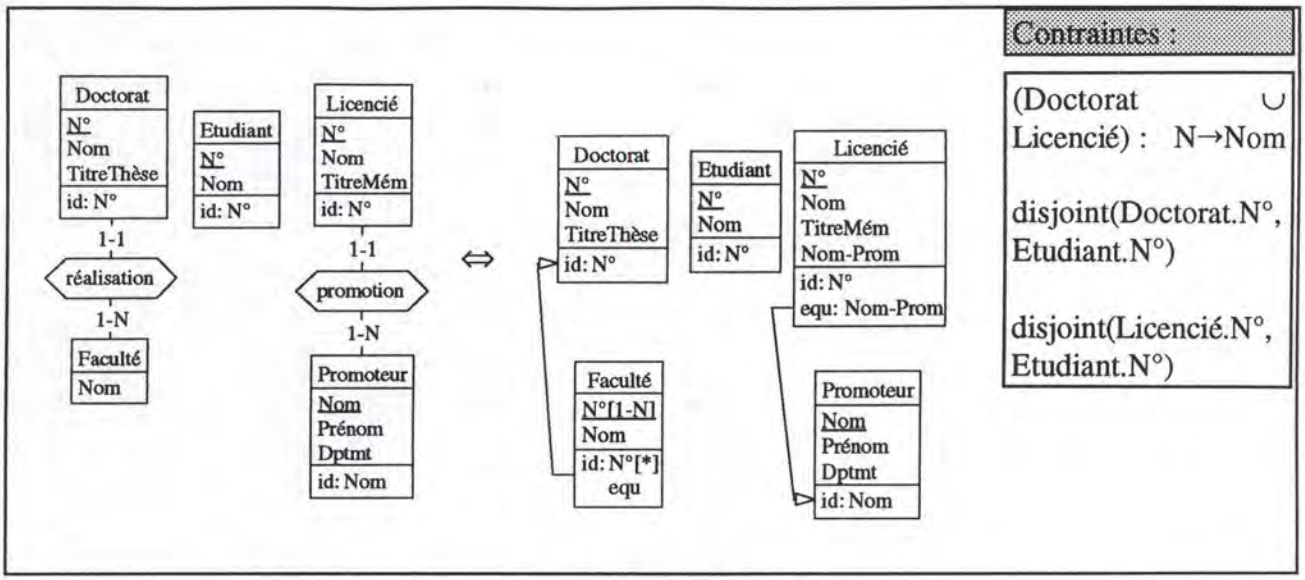


Figure 34 : Transformation des T.A. en clés étrangères

6.3.2 *Exemple 2*

Un second exemple montre un surtype participant à deux T.A. dont un T.A. implique aussi un sous-type.

Le surtype *Employé* possède un attribut, *Nom* qui n'est pas identifiant. *Employé* est impliqué dans un T.A. *contribution* avec le T.E. *Projet* et dans un T.A. *supervision* avec le T.E. *Manager* qui est aussi sous-type de *Employé*.

Manager possède un attribut : *Groupe_supervisé* multivalué et facultatif. Deux autres sous-types sont également présents dans cette relation is-a soumise à une contrainte de partition : le T.E. *Secrétaire*, avec un attribut multivalué *Capacité* et le T.E. *Ingénieur* avec l'attribut *Spécialité*.

De plus le T.E. *Secrétaire* est impliqué dans un T.A. utilisation où un T.E. *Ttt_texte* joue un rôle. La représentation de cet énoncé informel se trouve à la Figure 35.

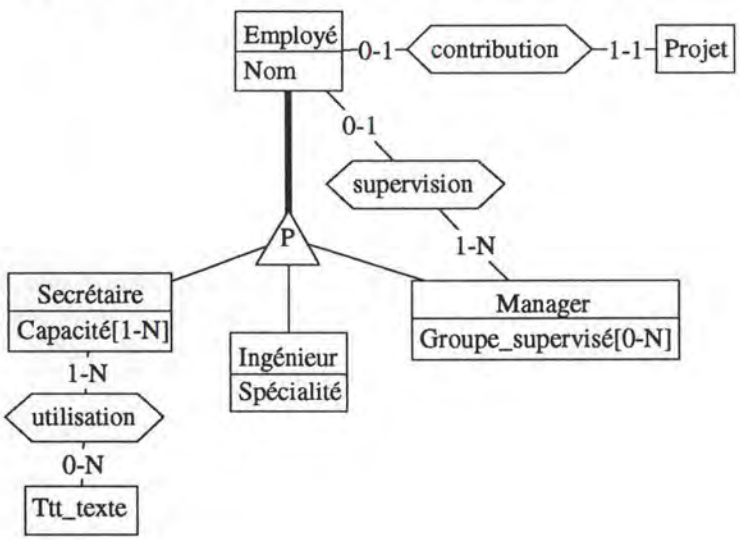


Figure 35 : Exemple de relation is-a (2)

Transformation par matérialisation

Si l'on désire une transformation par matérialisation, il est nécessaire d'introduire un identifiant technique dans le surtype pour pouvoir transformer les T.A. créés par la transformation de la relation is-a en clés étrangères.

Le résultat est donné à la Figure 36.

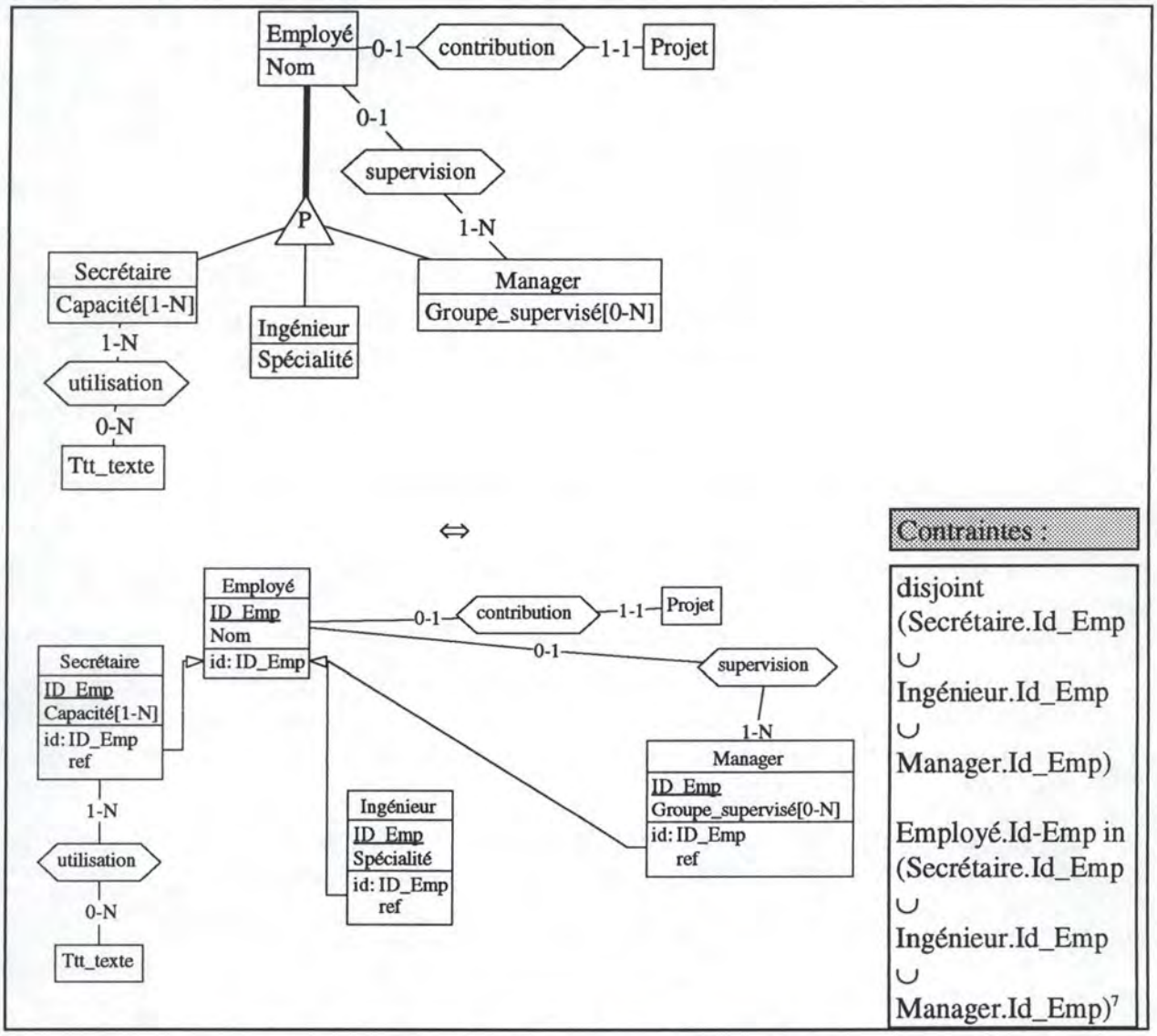


Figure 36 : Transformation par matérialisation simple

Pour rendre le schéma relationnel, il est nécessaire de transformer les T.A. Le T.A. many-to-many *utilisation* doit être transformée en T.E. Pour transformer un T.A., il faut un identifiant dans au moins un des T.E. sur lesquels porte le T.A. Nous avons choisi de mettre un identifiant dans le T.E. *Ttt-texte*

[7] Le code correspondant à ces contraintes peut être déterminé à partir du paragraphe 3.2

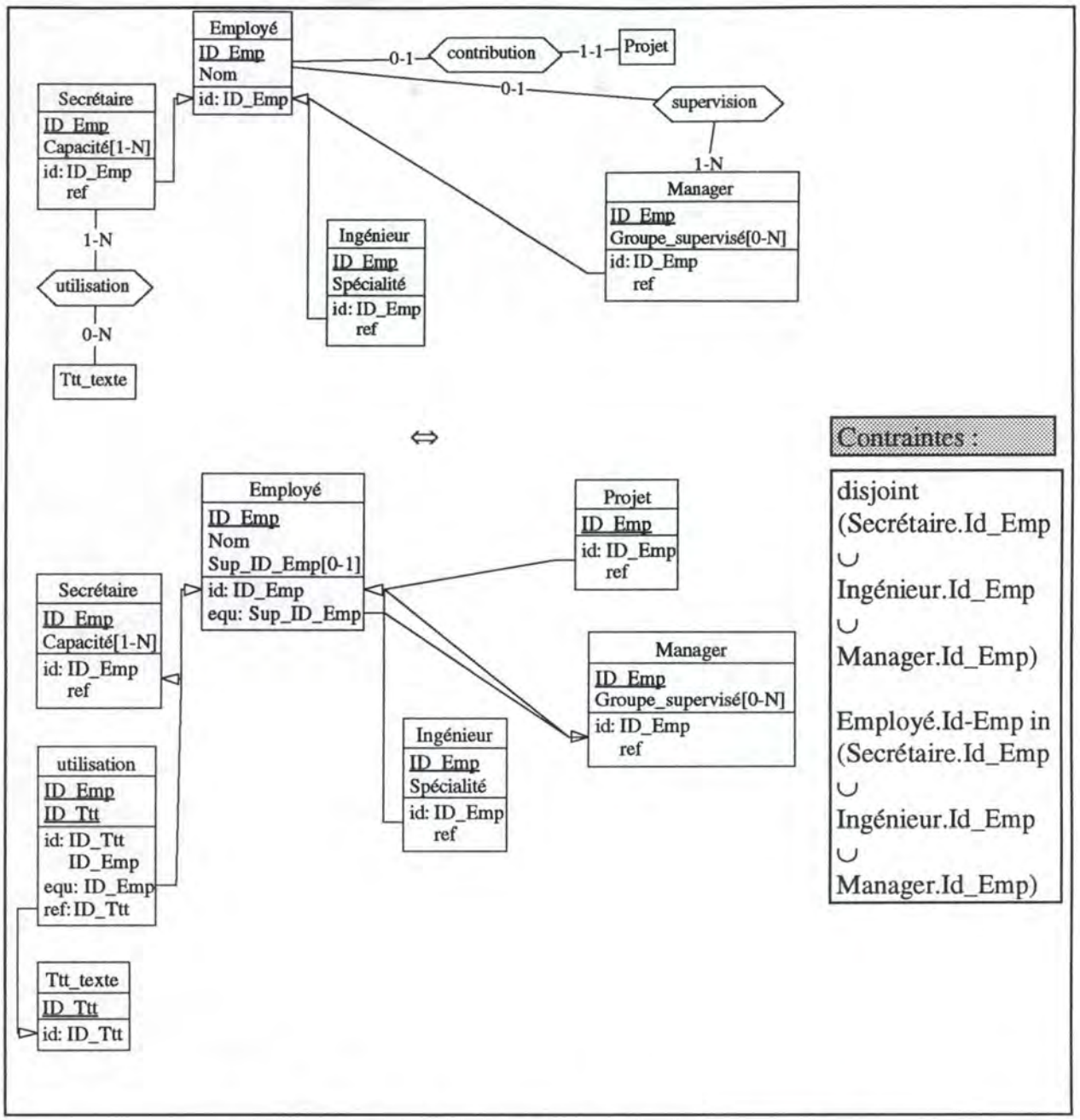


Figure 37 : Transformation des T.A. en clés étrangères

Remarquons que pour rendre le schéma relationnel, il est nécessaire de transformer les attributs multivalués *Capacité* et *Groupe_supervisé.* Les différentes méthodes sont d'en faire un T.E. ou de concaténer les valeurs.

Pour la méthode de l'indicateur de sous-types, il faut ajouter un identifiant technique au surtype avant de faire la transformation. Un attribut *TYPE* est créé dans le surtype et sa cardinalité vaut [1-1] (Figure 38).

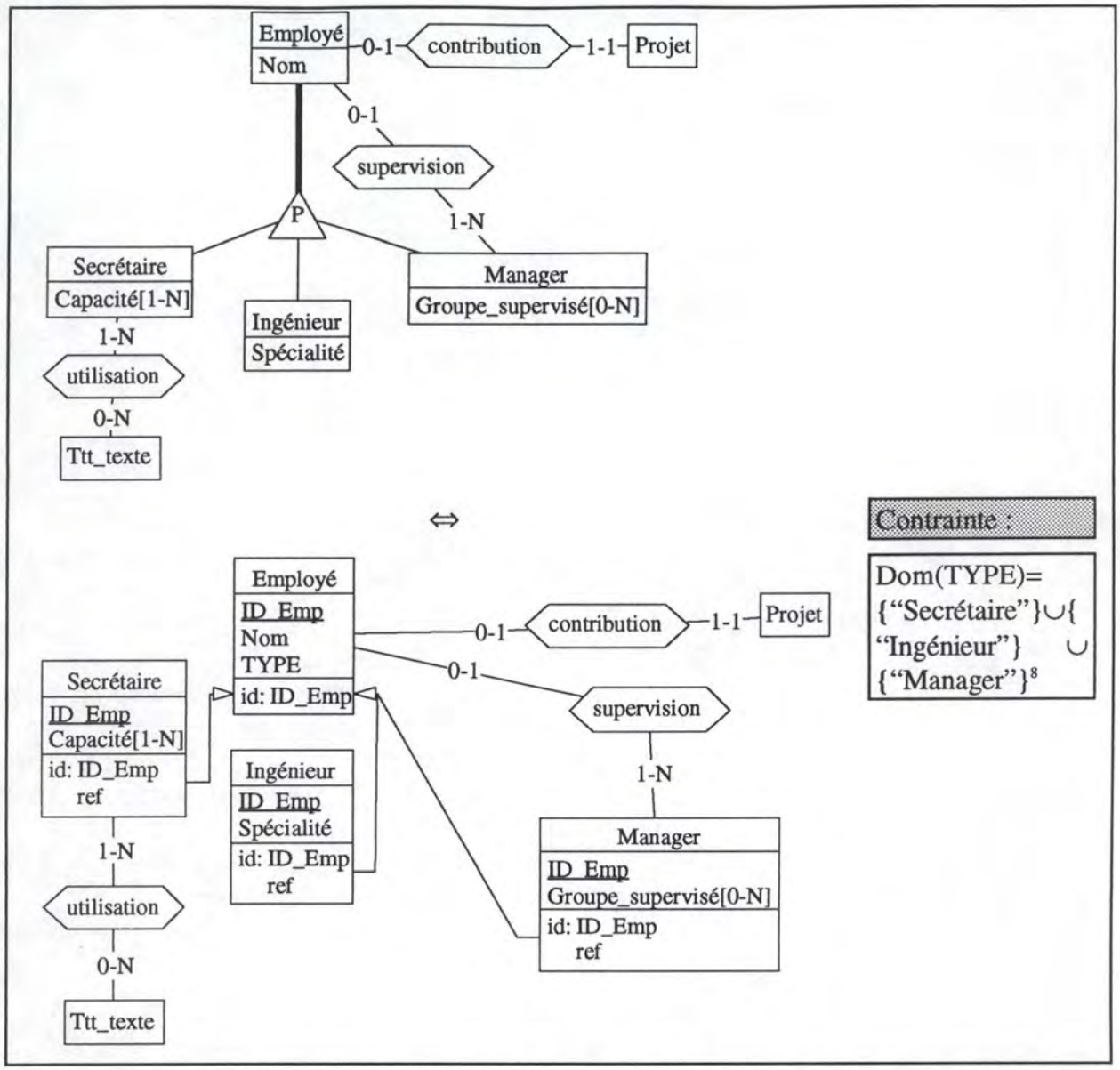


Figure 38 : Transformation par matérialisation par la méthode de l'indicateur de sous-types.

Il est nécessaire de transformer les T.A. many-to-many puis les T.A. one-to-many en clés étrangères, comme présenté plus haut.

[8] Le code correspondant à ces contraintes peut être déterminé à partir du paragraphe 3.2

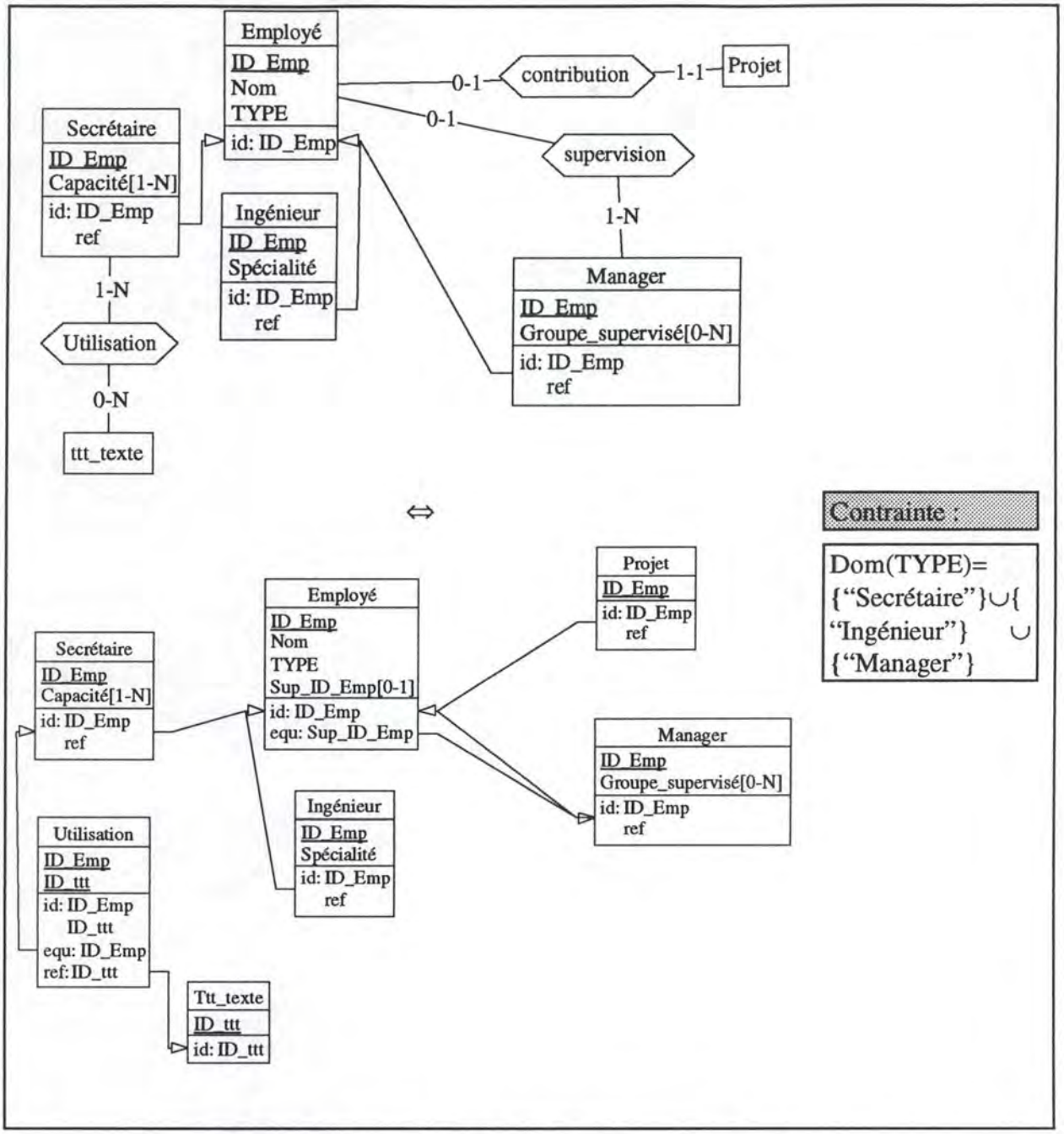


Figure 39 : Transformation des T.A. en clés étrangères

Les attributs *Capacité* et *Groupe_supervisé* doivent être transformés

Transformation par héritage ascendant

Si l'on veut faire la transformation de la relation is-a par héritage ascendant, il n'est pas nécessaire d'ajouter un identifiant technique à *Employé*. Le résultat de cette transformation est donné à la Figure 40. Le T.E. *Employé* joue toujours le rôle *est supervisé* par mais il joue aussi le rôle *supervise* dans le même T.A. On appelle un tel type d'associations, un T.A. cyclique.

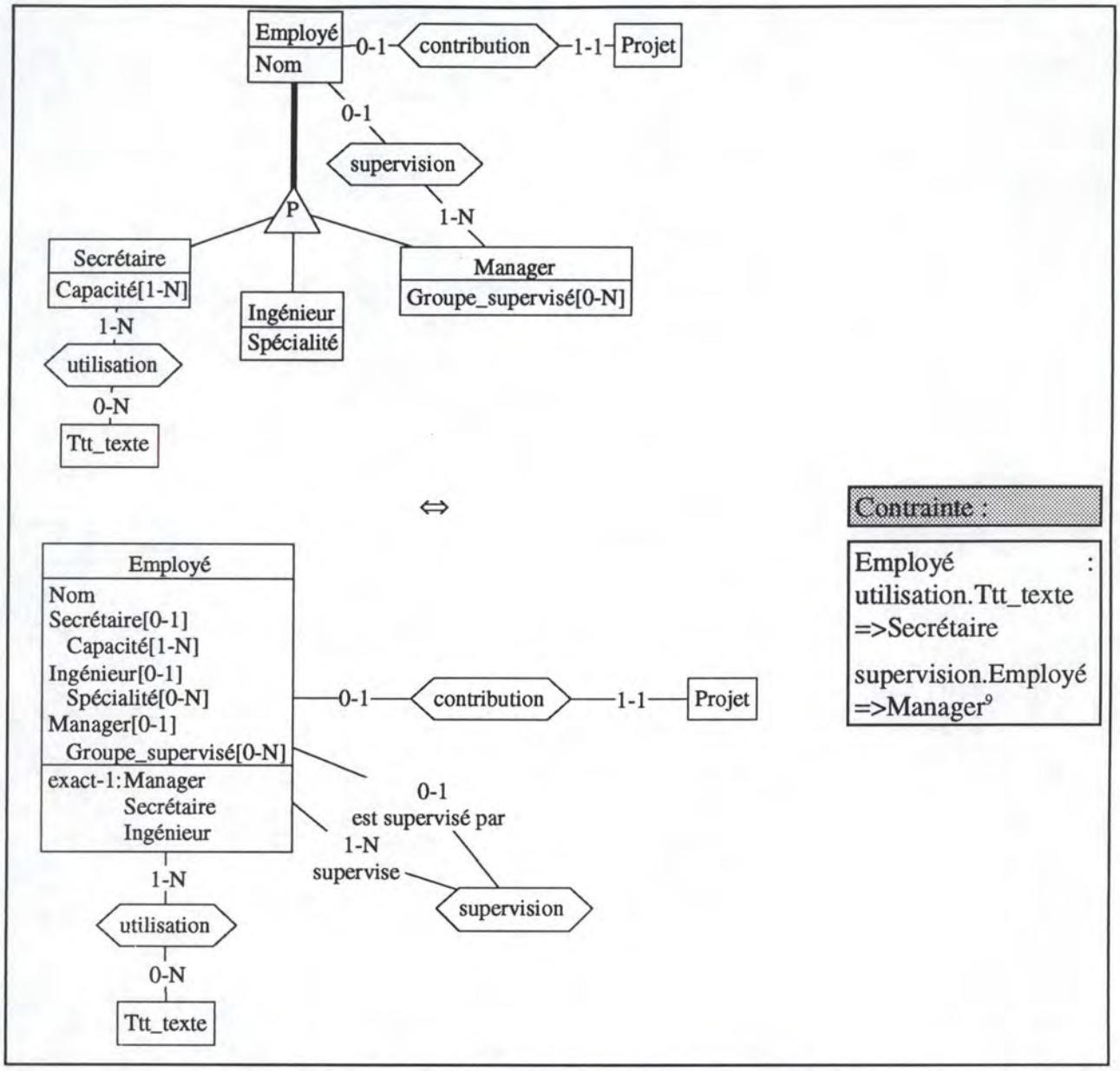


Figure 40 : Transformation par héritage ascendant

Les T.A. doivent être transformés. Pour permettre la transformation du T.A. *contribution,* il est nécessaire d'avoir un identifiant dans un des T.E. sur lesquels le T.A. porte. Nous avons choisi de mettre un identfiant technique dans le T.E. *Employé*. Le T.A. cyclique est transformé en une clé étrangère ayant pour origine et cible le T.E. *Employé*. La Figure 41 présente les résultats de telles transformations.

[9]Le code correspondant à ces contraintes peut être déterminé à partir du paragraphe 4.5

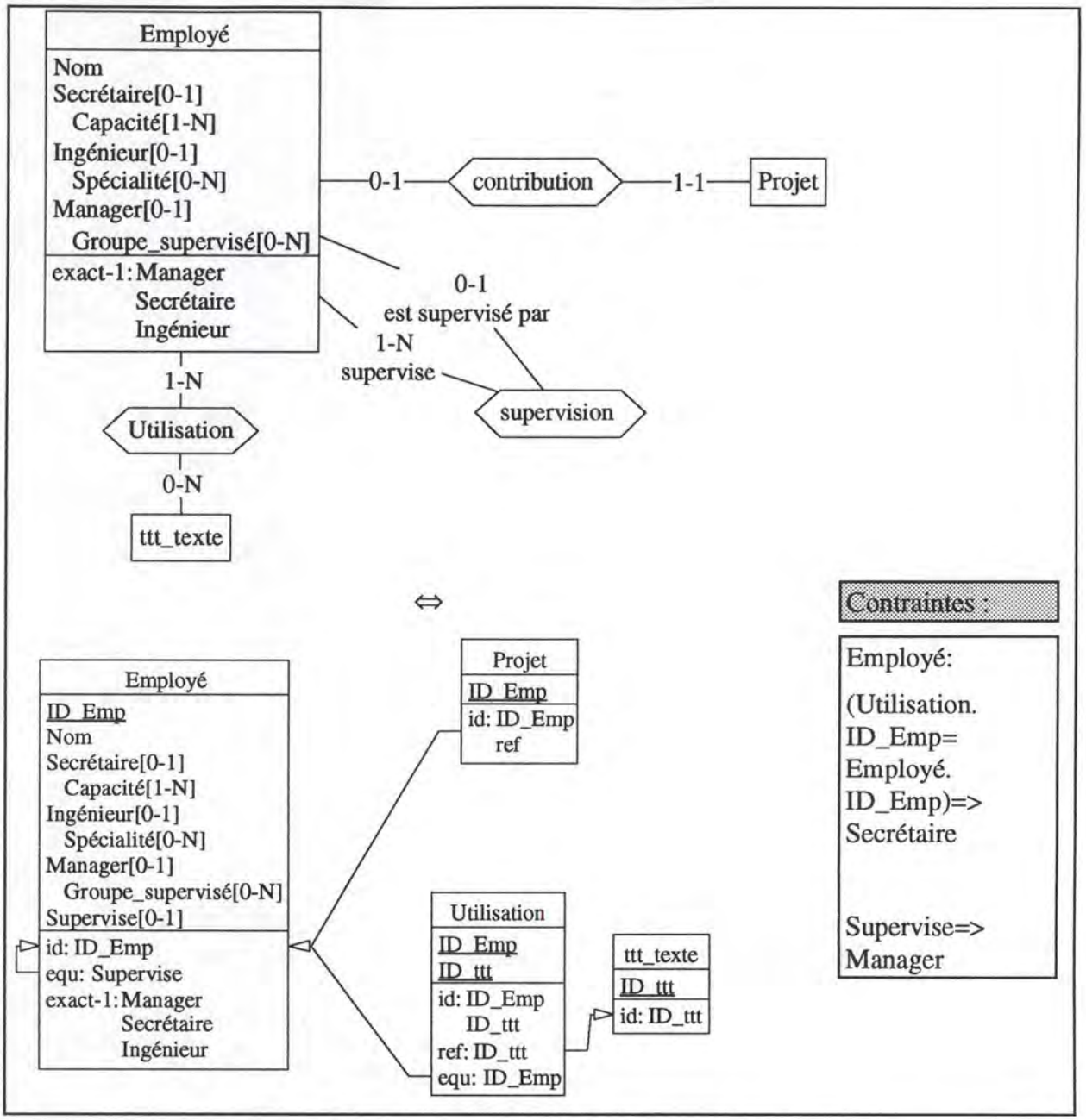


Figure 41 : Transformation des T.A. en clés étrangères

Il est nécessaire de transformer les attributs décomposables et les attributs multivalués.

Transformation par héritage descendant

La transformation par héritage descendant nécessite l'ajout d'un identifiant technique dans le surtype puis une transformation des T.A.

Dans notre cas, la transformation par héritage descendant crée 3 associations contribution (*contr_1*, *contr_2*, *contr_3*) de *Projet* vers chacun des sous-types. Il en est de même pour l'autre T.A. portant sur le surtype, *supervision*. Les T.A. sont créés entre le sous-type *Manager* et les deux autres sous-types mais également de *Manager* vers lui-même (T.A. cyclique). Le résultat de la transformation se trouve à la Figure 42. Il reste à transformer les T.A. en clés étrangères.

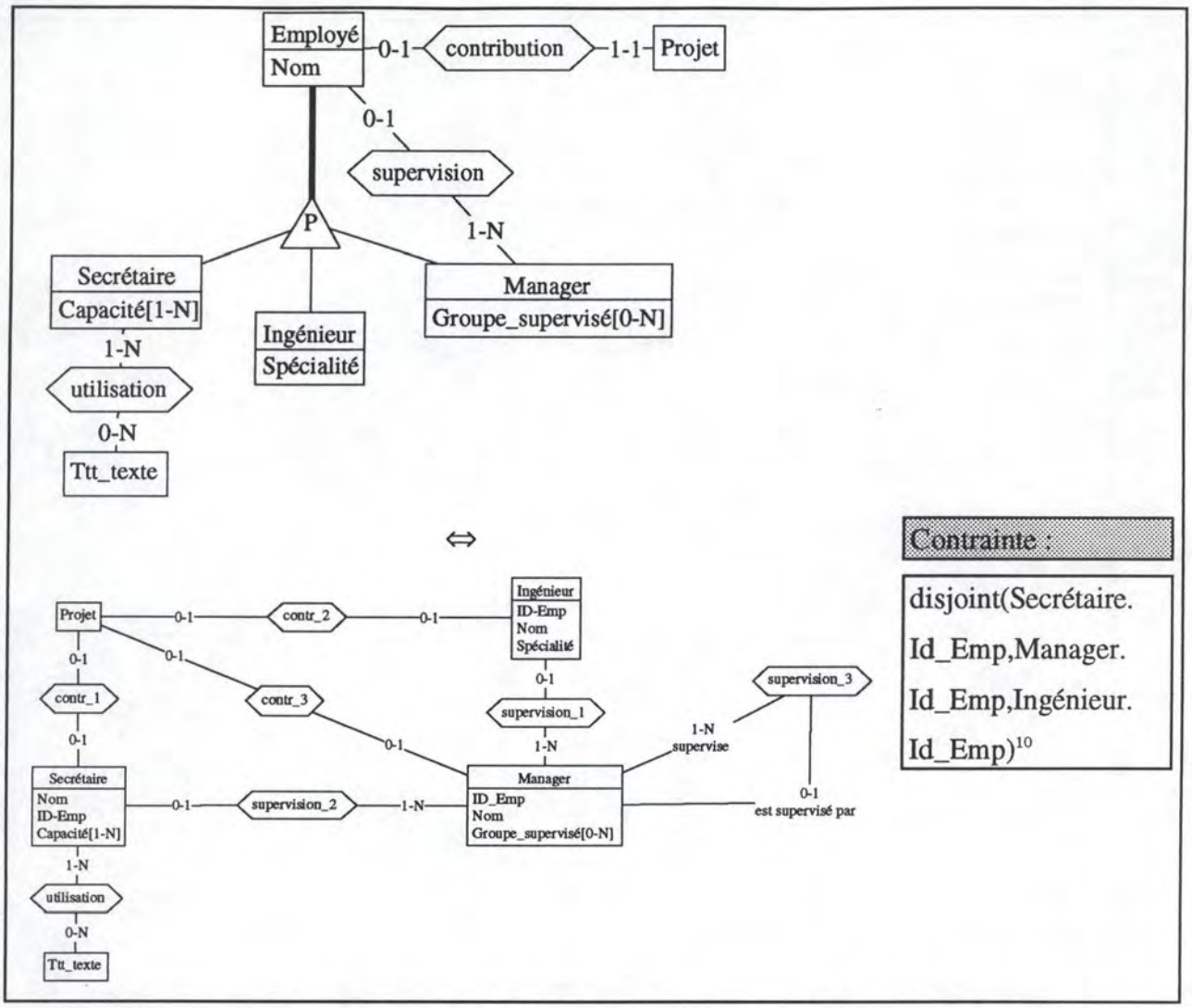


Figure 42 : Transformation de la relation is-a par héritage descendant.

Il faut transformer les T.A. en clés étrangères. Pour cela, certains identifiants techniques doivent être ajoutés dans les T.E. C'est le cas des T.E. *Projet, Secrétaire, Manager* et *Ttt_texte*. Le résultat des transformations est présenté à la Figure 43 .

[10] Le code correspondant à cette contrainte peut être déterminé à partir du paragraphe 5.2

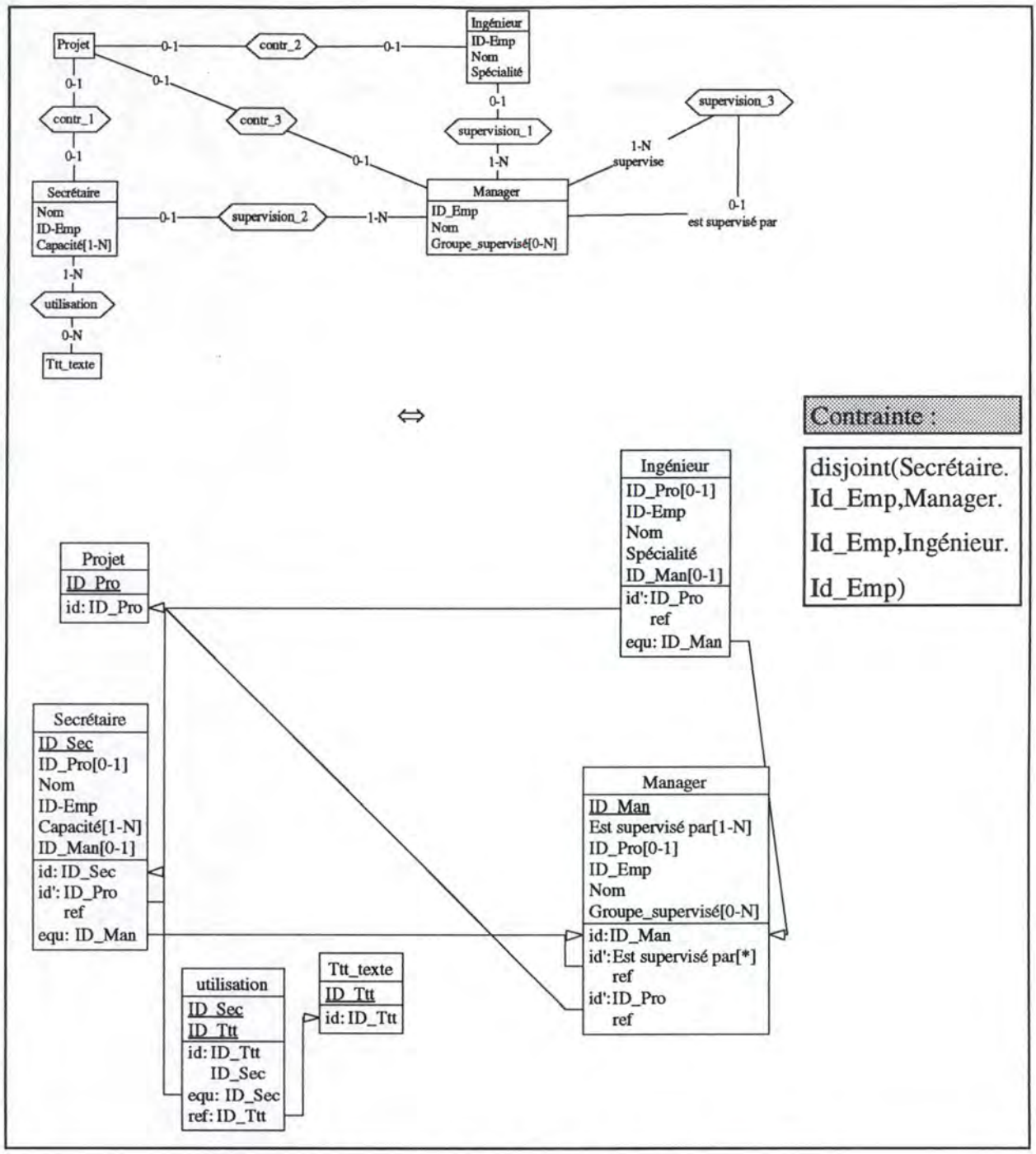


Figure 43 : Transformation des T.A. en clés étrangères

Pour rendre le schéma relationnel, il est nécessaire de transformer les attributs multivalués.

6.4 Conclusion

A travers cette étude, nous avons montré que le problème est plus complexe que ce qui n'est généralement repris dans la littérature. Celle-ci ne propose généralement que des traductions trop faibles. Cette étude va nous permettre de traduire des relations is-a dans DB-Main de manière correcte et complète. Cet outil permet la transformation de relation is-a et la génération du code SQL comprenant également les contraintes traduites.

III.TRANSFORMATIONS POSSIBLES DANS DB_MAIN[10],[12]

Introduction

L'atelier DB-Main ne propose pas toutes les transformations de relations is-a. Ce chapitre présente les différentes possibilités de DB-Main pour transformer les relations is-a et pour générer le code SQL résultant des transformations.

DB-Main propose pour la transformation de schémas un menu d'assistants de transformation dédiés à des classes de problèmes spécifiques. De plus, DB-Main possède un générateur de code SQL : il prend le schéma courant et génère le code SQL correspondant. Pour améliorer le générateur SQL de l'atelier, nous désirons permettre la génération de contraintes pour compléter le schéma relationnel de relations is-a transformées.

Le but de ce chapitre est de donner les différentes améliorations à apporter à l'atelier pour que les transformations soient complètes et correctes et que le code SQL soit complet.

1. DB-Main[6],[7],[9],[14]

Jusqu'à présent, seules les transformations de matérialisation et d'héritage ascendant sont prises en compte dans l'atelier. Dans ces transformations, les relations is-a peuvent être ou non soumises à des contraintes de disjonction, de totalité ou de partition.

1.1 Transformations de matérialisation

La transformation par matérialisation est prise en compte par l'atelier mais il manque des contraintes. Par contre, la transformation par création d'un indicateur de sous-type n'est pas encore prévue de manière automatique.

Lors de la transformation de relations is-a avec création d'attributs dans le surtype, comme dans le cas de la figure 9 du chapitre 2, le code SQL généré par l'atelier ne reprend pas la contrainte (Figure 1).

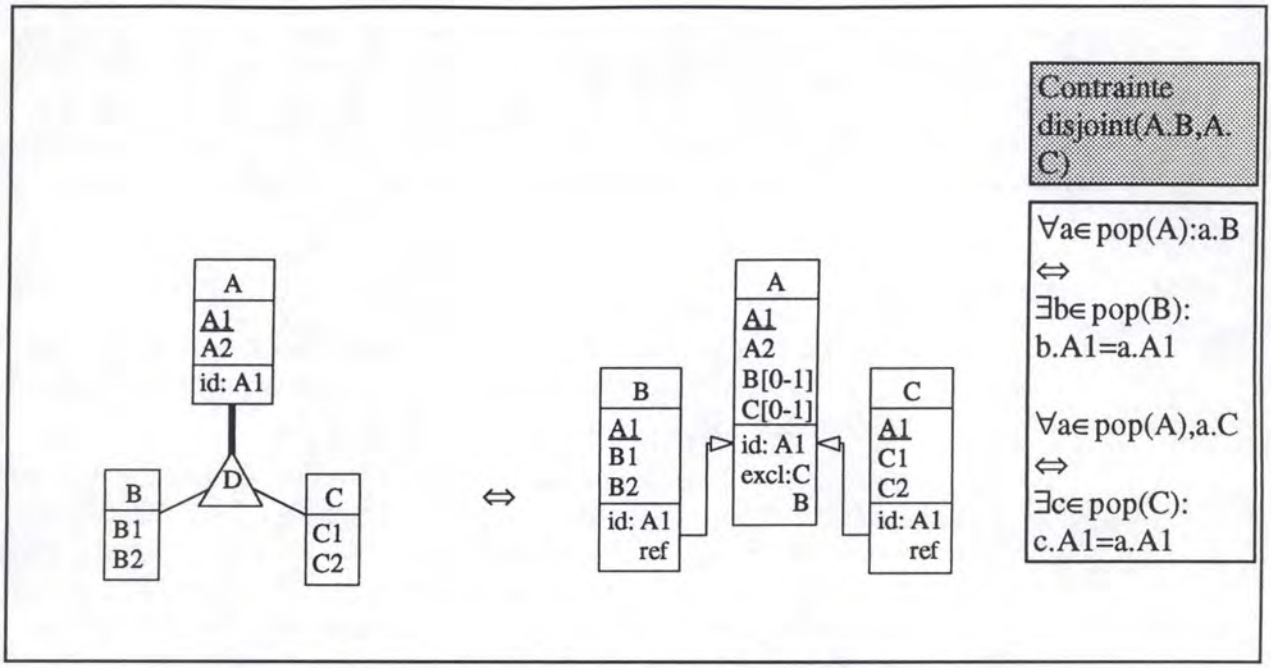


Figure 1 : Transformation par création de clés étrangères

Nous avons choisi de ne pas développer cette méthode mais plutôt une alternative où les contraintes d'exclusion et de at-lst-one sont exprimées en check qui vérifie la disjonction et/ou la totalité entre les sous-types.

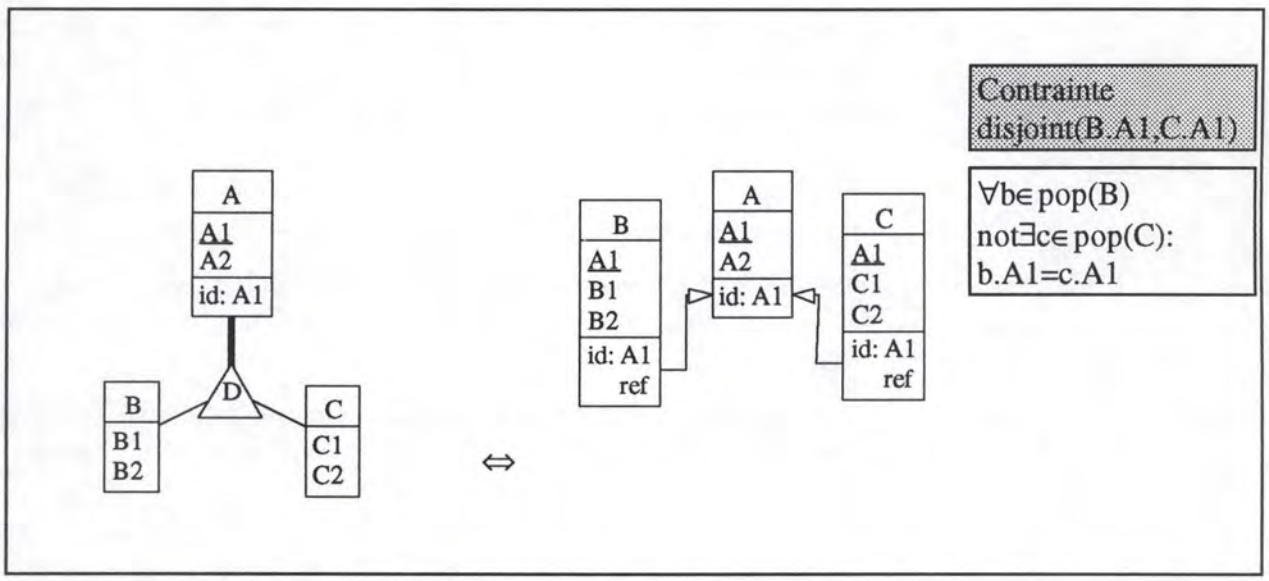


Figure 2 : Transformation par création de clés étrangères avec check

Le code SQL correspondant à une contrainte de disjonction est :

```
alter table B add constraint DIS_B
check(A1 not in
        (select A1 from C)
        and ...
        and (A1 not in
         (select A1 from Z)) ;

alter table C add constraint DIS_C
check(A1 not in
        (select A1 from B)
        and ...
       and (A1 not in
       (select A1 from Z)) ;

...

alter table Z add constraint DIS_Z
check(A1 not in
        (select A1 from B)
         and ...
         and (A1 not in
         (select A1 from C)) ;
```

Pour la contrainte de totalité, le code est :

```
alter table A
check(A1 in
        (select A1 from B))
         or (A1 in
         (select A1 from C))
         or...
         or (A1 in
         (select A1 from Z)) ;
```

Pour la transformation par la technique de l'indicateur de sous-type, un attribut TYPE est créé. Des contraintes viennent compléter le schéma :

```
alter table B
check (
   not exists(A.TYPE
   where A.A1 = B.A1
   and TYPE not like %"B%" ));

alter table C
check (
   not exists(A.TYPE
   where A.A1 = C.A1
   and TYPE no like %"C% "));

...

alter table Z
check (
   not exists(A.TYPE
   where A.A1 = Z.A1
   and TYPE not like %"Z"%));
```

Pour remplir cet attribut TYPE avec les noms des sous-types, nous avons choisi un trigger.

Ce trigger vient mettre, à chaque insertion de données dans un sous-type, le nom de ce sous-type dans l'attribut TYPE du surtype. Si par exemple, il s'agit d'une relation is-a subissant une contrainte de disjonction et que TYPE comprend déjà une valeur, l'insertion du second sous-type sera interdite par la cardinalité de l'attribut TYPE.

Pour les relations is-a non soumises à une contrainte de disjonction, le trigger est :

```
create trigger TRG_TYPE for B
before insert as
begin
        update A set TYPE = TYPE || '"B"'
        where A.A1 = NEW.A1
end ;
```

Un trigger semblable est créé pour l'introduction de données dans les autres sous-types.

Notons que le symbole || indique ici une concaténation de deux strings.

Pour les relations is-a soumises à une contrainte de disjonction, il faut tester si l'attribut TYPE possède déjà une valeur avant d'introduire une autre valeur.

Remarquons qu'il est nécessaire d'utiliser une transaction pour introduire les données. Cette transaction vient mettre dans l'attribut TYPE du surtype correspondant, le nom du sous-type entre double guillemets.

1.2 Transformation par héritage ascendant

Pour réaliser une transformation par la technique d'héritage ascendant dans l'atelier, il faut passer par une transformation par matérialisation puis par une transformation des T.E. sous-types en des attributs facultatifs et décomposables.

La transformation de relation is-a dont un sous-type joue un rôle dans un T.A. R est également prise en compte mais il faut passer par une transformation du type d'associations. R en clés étrangères avant de pouvoir remonter les T.E. sous-types dans le surtype.

1.3 Transformation par héritage descendant

La technique de transformation par héritage descendant n'est pas du tout prise en compte par DB-Main.

Il faut dès lors programmer toutes les transformations étudiées dans le chapitre précédent et permettre la génération du code SQL correspondant.

1.4 Conclusion

En conclusion, le Tableau 1 reprend certaines relations is-a que l'on peut rencontrer dans le cas d'héritage simple et en colonnes, les différentes techniques possibles déjà prévues ou à prévoir dans l'atelier. A l'intersection d'une ligne et d'une colonne se trouvent les éléments qui manquent dans l'atelier pour que la génération ou la transformation soit correcte et complète. Si la transformation est impossible ou n'est que partiellement possible, la case est grisée.

	Transfo. par matérialisation		**Transfo. par héritage ascendant**	**Transfo. par héritage descendant**
	simple	***création indicateur de sous-types***		
relation is-a sans identifiant dans le surtype				particulièrement possible (si les sous-types sont disjoints) : transformation et génération d'une contrainte
relation is-a avec un identifiant dans le surtype	transformation(sans création d'attributs dans le surtype) et génération d'une contrainte	transformation et génération d'une contrainte		transformation et génération d'une contrainte
relation is-a dont un sous-type joue un rôle dans un T.A.	transformation(sans création d'attributs dans le surtype) et génération d'une contrainte	transformation et génération d'une contrainte	transformation et génération d'une contrainte	transformation et génération d'une contrainte

Tableau 1: Transformation possibles prévues et non prévues dans l'atelier

IV.MODIFICATION DU GENERATEUR SQL[6],[7],[10],[12],[14]

Introduction

Pour pouvoir programmer la génération des contraintes vues dans les chapitres précédents, nous avons besoin de retrouver certains composants du schéma courant, comme les sous-types, le surtype,... Pour cela, nous proposons de mettre en évidence au moment des transformations, les éléments nécessaires à la génération du code SQL des contraintes en *Voyager 2*.

1. Transformation par matérialisation

1.1 Transformation simple

Dans la transformation par matérialisation simple, seules les relations is-a dont le surtype est soumis à des contraintes de disjonction, de totalité et de partition donnent après transformation des contraintes à exprimer en SQL.

Les Figure 1, Figure 2 et Figure 3 reprennent les transformations et les contraintes pour des relations is-a ayant un identifiant dans le surtype car, comme vu précédemment, l'identifiant dans le surtype est un des éléments nécessaires pour faire la transformation par matérialisation.

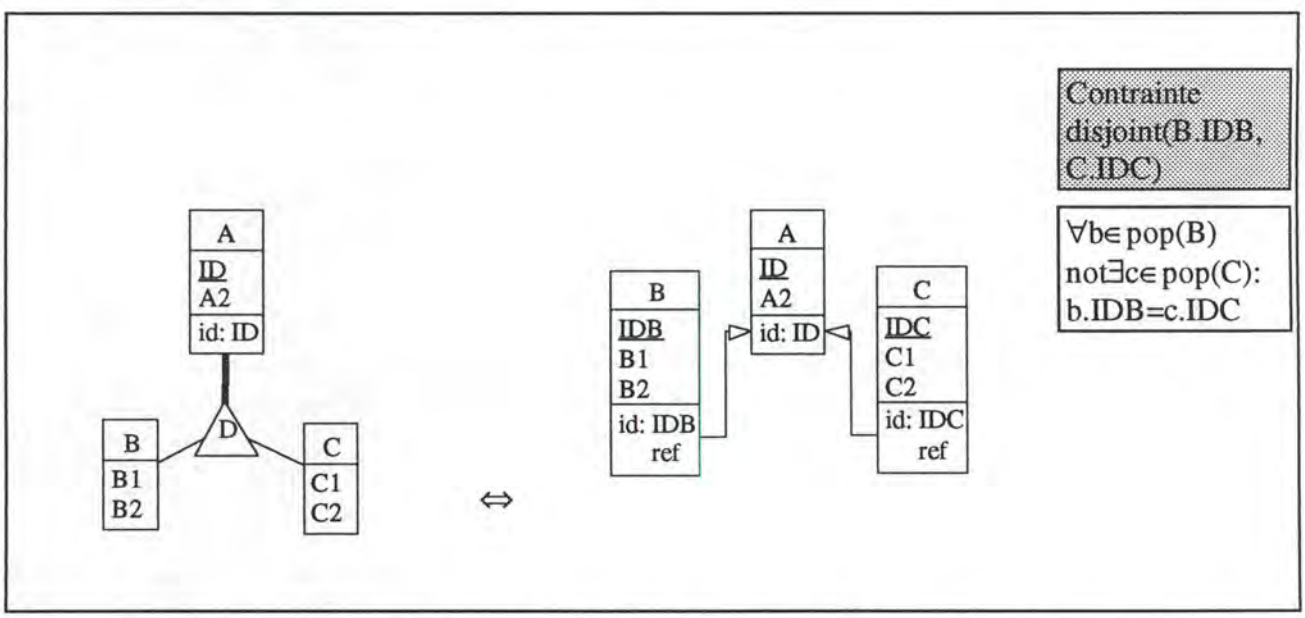


Figure 1 : Transformation par matérialisation de relation is-a soumises à une contrainte de disjonction

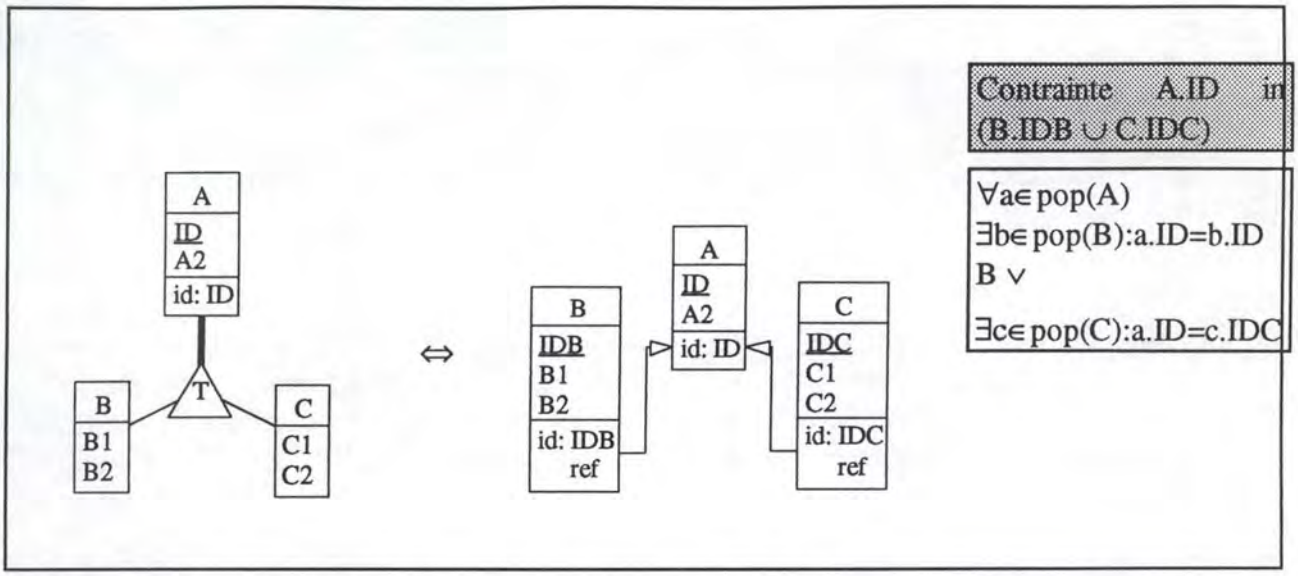


Figure 2 : Transformation par matérialisation de relation is-a soumises à une contrainte de totalité

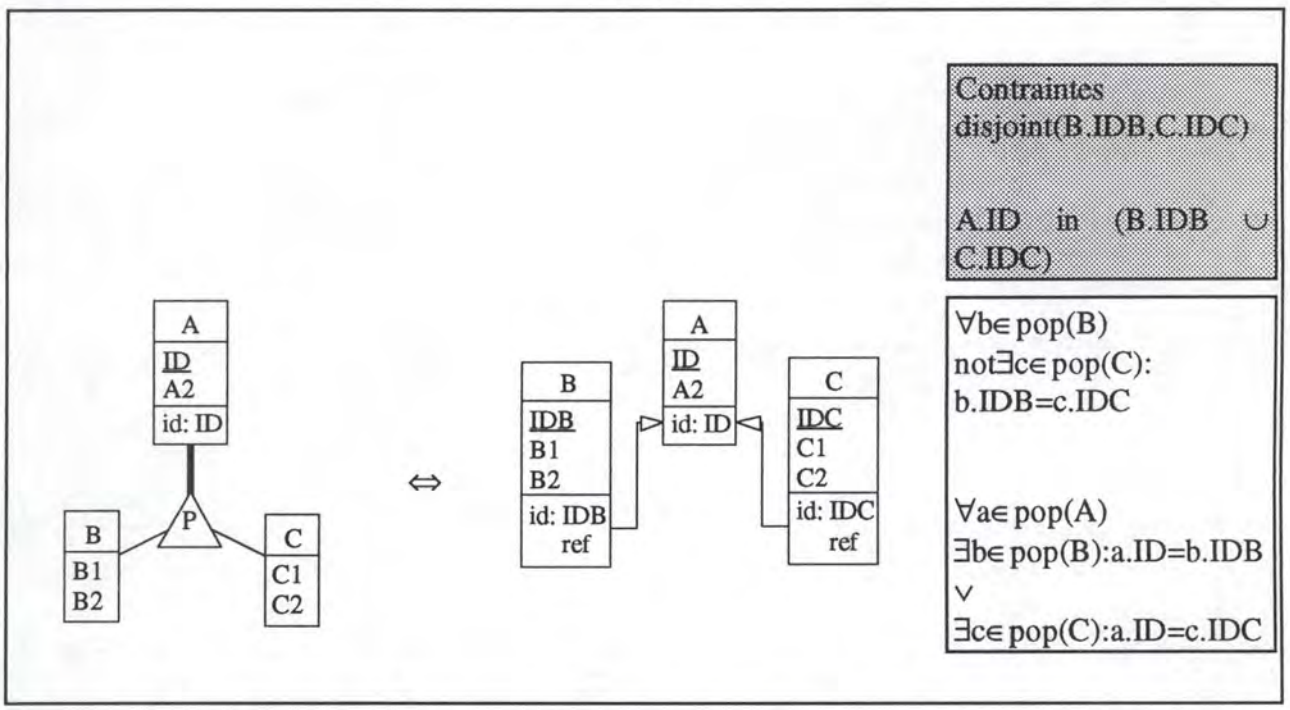


Figure 3 : Transformation par matérialisation de relations is-a soumises à une contrainte de partition

1.1.1 Code SQL

Si les transformations vues dans les Figure 1, Figure 2 et Figure 3 étaient prévues par l'atelier, le générateur SQL donnerait le code correct pour la génération des tables A, B et C. Il reste à générer les contraintes.

La contrainte disjoint(B.IDB,C.IDC) s'exprime en SQL par

```
alter table B add constraint DIS_B
```

```
check(IDB not in
(select IDC from C)) ;

alter table C add constraint DIS_C
check(IDC not in
        (select IDB from B)) ;
```

La contrainte pour la totalité

```
alter table A add constraint TOT_A
check(ID in
        (select IDB from B))
        or (ID in
        (select IDC from C))
```

La contrainte pour la partition reprend les deux codes précédents.

1.1.2 Programmation

Pour pouvoir exprimer ces contraintes dans le schéma, nous allons réaliser une transformation dans DB-Main qui met dans SEM du surtype : #supertype=D pour une contrainte de disjonction, # supertype =T pour une contrainte de totalité et # supertype =P pour une contrainte de partition. Pour la relation is-a sans contrainte, SEM contient # supertype = . La transformation place dans SEM de la foreign key des sous-types référençant le surtype, #subtype=. Une fonction nous permet de retrouver les sous-types d'un surtype; une autre de retrouver l'identifiant d'un T.E. Il ne reste donc qu'à générer le code SQL de la contrainte.

Le code Voyager correspondant est repris dans l'Annexe A

1.2 Transformation par création d'attributs booléens

Dans l'atelier DB-Main, la transformation précédenten'est pas prévue. Par contre, nous pouvons y trouver la transformation où des attributs booléens sont créés. La contrainte d'exclusion, at-lst-1 ou exactly-1 est exprimée entre ces attributs (Figure 4).

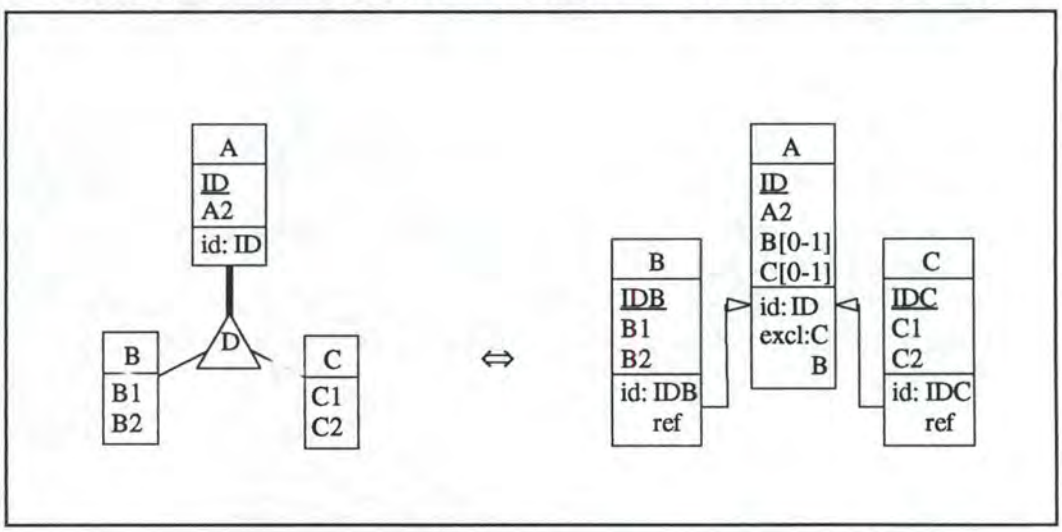


Figure 4 : Transformation prévue dans l'atelier

1.2.1 Code SQL

Le code SQL de la contrainte d'exclusion, généré par DB-Main, vaut :

```
alter table A add constraint ISAA
check ((C is not null and B is null)
    or (C is null and B is not null)
    or (C is null and B is null));
```

Pour la contrainte at-lst-1, l'atelier génère le code suivant

```
alter table A add constraint ISAA
check (C is not null or B is not null);
```

Pour la contrainte exact-1, le code généré par l'atelier est :

```
alter table A add constraint ISAA
check ((C is not null and B is null)
    or (C is null and B is not null));
```

Pour cette transformation, un trigger est nécesssaire pour introduire dans B (respectivement C) une valeur quelconque, choisie comme étant *, lorsque l'on introduit des valeurs dans le T.E. B (respectivement C).

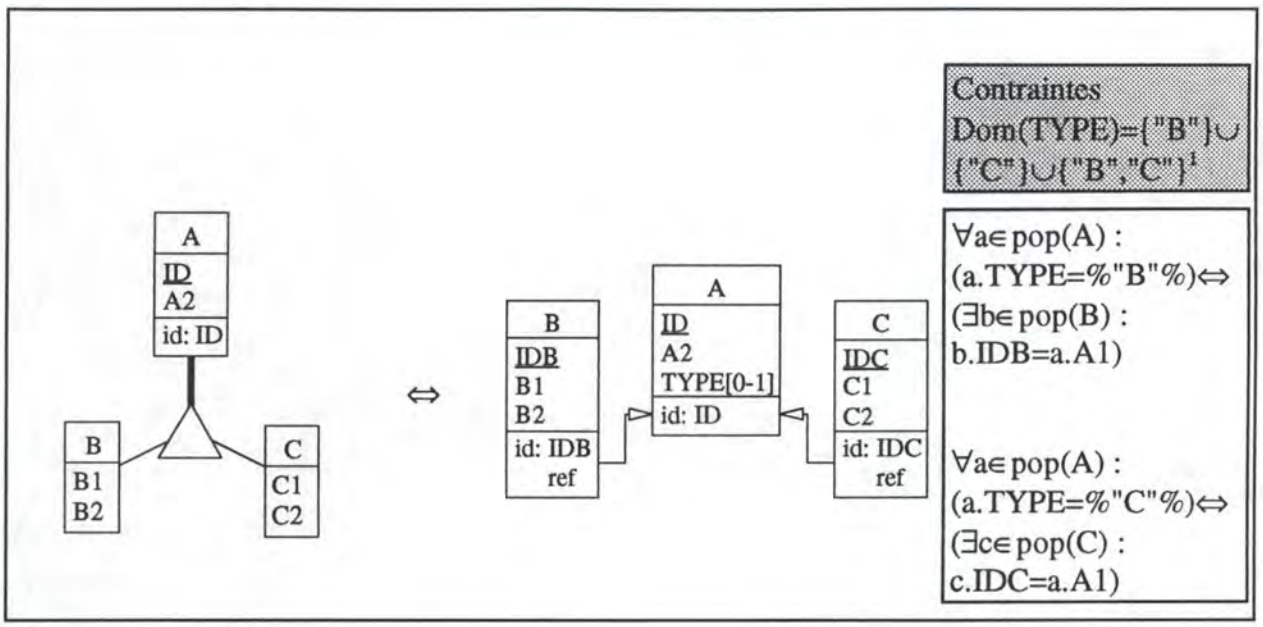
```
create trigger TRG_B for B
active before insert position 0 as
begin
update A set B = *
where A.ID = new.ID
end ;

create trigger TRG_C for C
active before insert position 0 as
begin
update A set C = *
where A.ID = new.ID
end ;
```

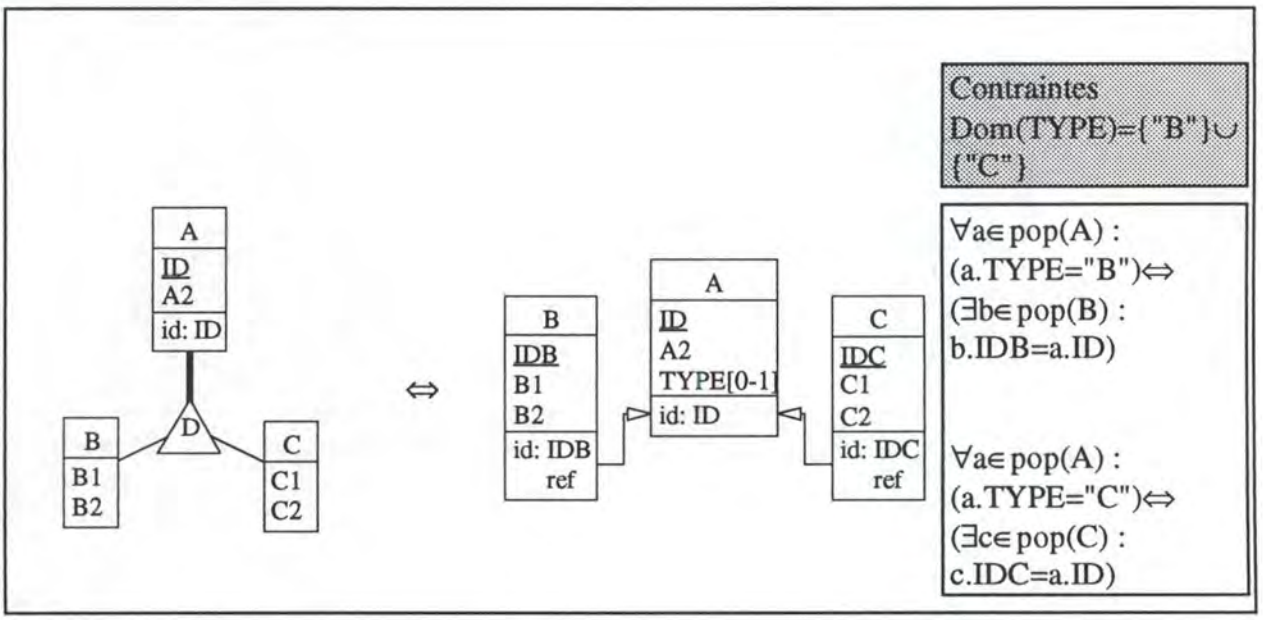
Nous devons à ce niveau faire une remarque. Dans le cas de la transformation prévue par l'atelier, pour introduire des valeurs dans les T.E., il est nécessaire d'utiliser une transaction.

Une transaction permet de faire passer la base de données d'un état consistent à un autre état consistent sans préserver la consistance aux points intermédiaires. Ici, la transaction nous permet d'introduire des valeurs dans les T.E. A, B et C sans se soucier des foreign keys. Le trigger ajoute ensuite dans l'attribut B ou/et C si nécessaire la valeur *.

1.3 *Transformation par la méthode d'indicateur de sous-types*

La transformation par la méthode de l'indicateur de sous-types donne les résultats suivants :





[1] Pour rappel, l'attribut TYPE est en fait un attribut multivalué représenté par un attribut monovalué dont la valeur est la concaténation des valeurs

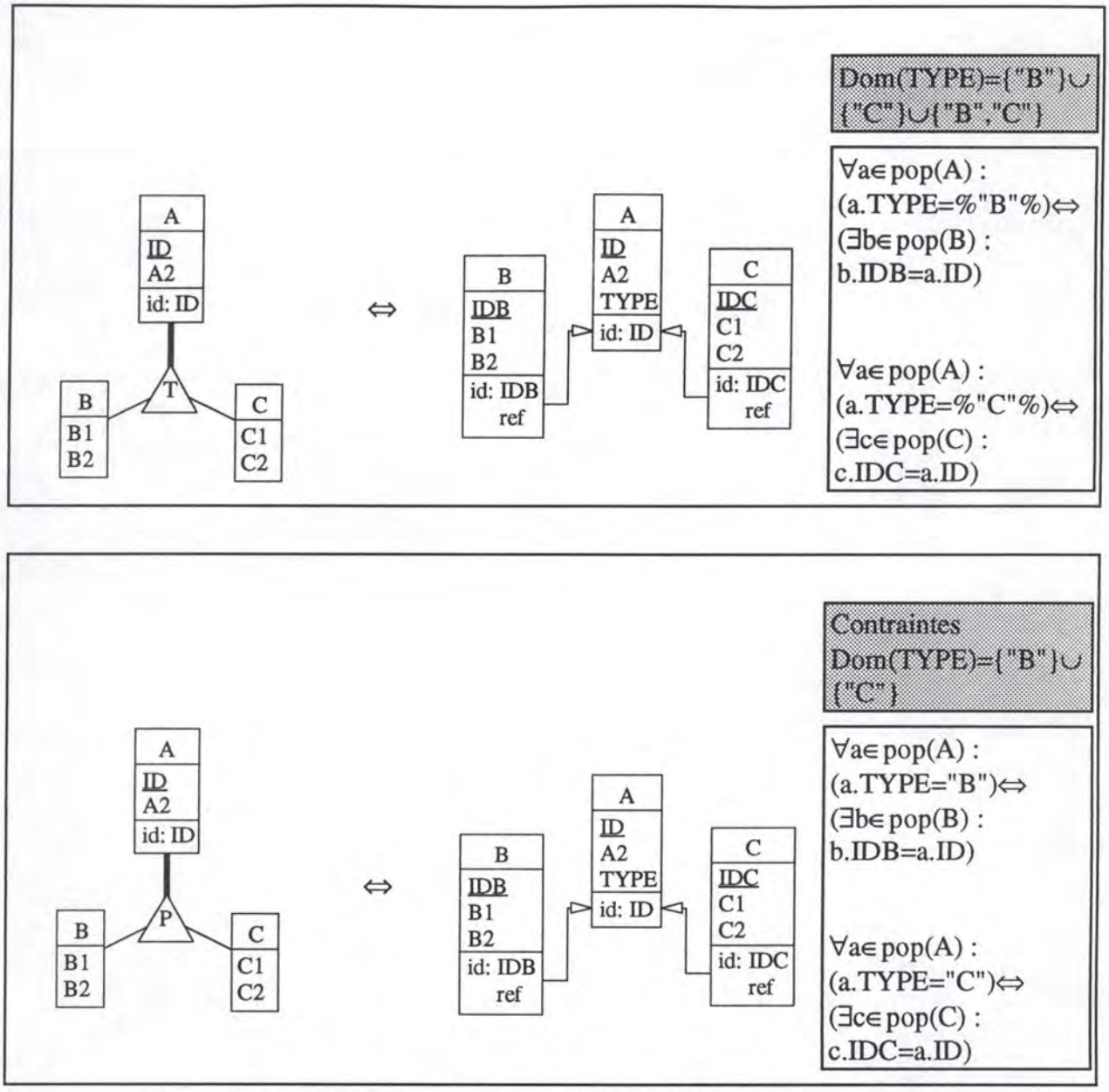


Figure 5 : Transformation par la méthode de l'indicateur de sous-types

1.3.1 Code SQL

Cette transformation n'est pas prévue dans l'atelier. Si elle l'était, l'atelier générerait correctement le code des tables A, B et C. Mais le code des contraintes resterait à générer.

```
alter table B add constraint TYPE_B
      check (not exists
        (select TYPE from A
         where A.ID = B.IDB
         and TYPE not like '%"B"%' ));

      alter table C add constraint TYPE_C
      check (not exists
        (select TYPE from A
         where A.ID = C.IDC
         and TYPE no like '%"C"%' ));
```

1.3.2 Programmation

Pour générer ces contraintes, nous avons choisi la technique suivante : mettre dans la description SEM de l'attribut TYPE une contrainte indiquant que c'est cet attribut qui contiendra les noms des sous-types auquel le surtype appartient.

La syntaxe de cette contrainte est simplement : #ISA_TYPE=.

Il faut donc parcourir tous les attributs du surtype et rechercher celui qui possède une contrainte #ISA_TYPE dans sa description SEM. Avec le nom des sous-types et celui du surtype, on peut programmer la génération de la contrainte.

Le code Voyager de cette génération se trouve dans l'annexe B

Dans ce cas également, il est nécessaire pour introduire les données dans les T.E. A, B et C d'utiliser un trigger pour mettre dans TYPE le nom du sous-type dans lequel on introduit les données.

2. *Transformation par héritage ascendant*

La transformation par héritage ascendant donne les résultats suivants :

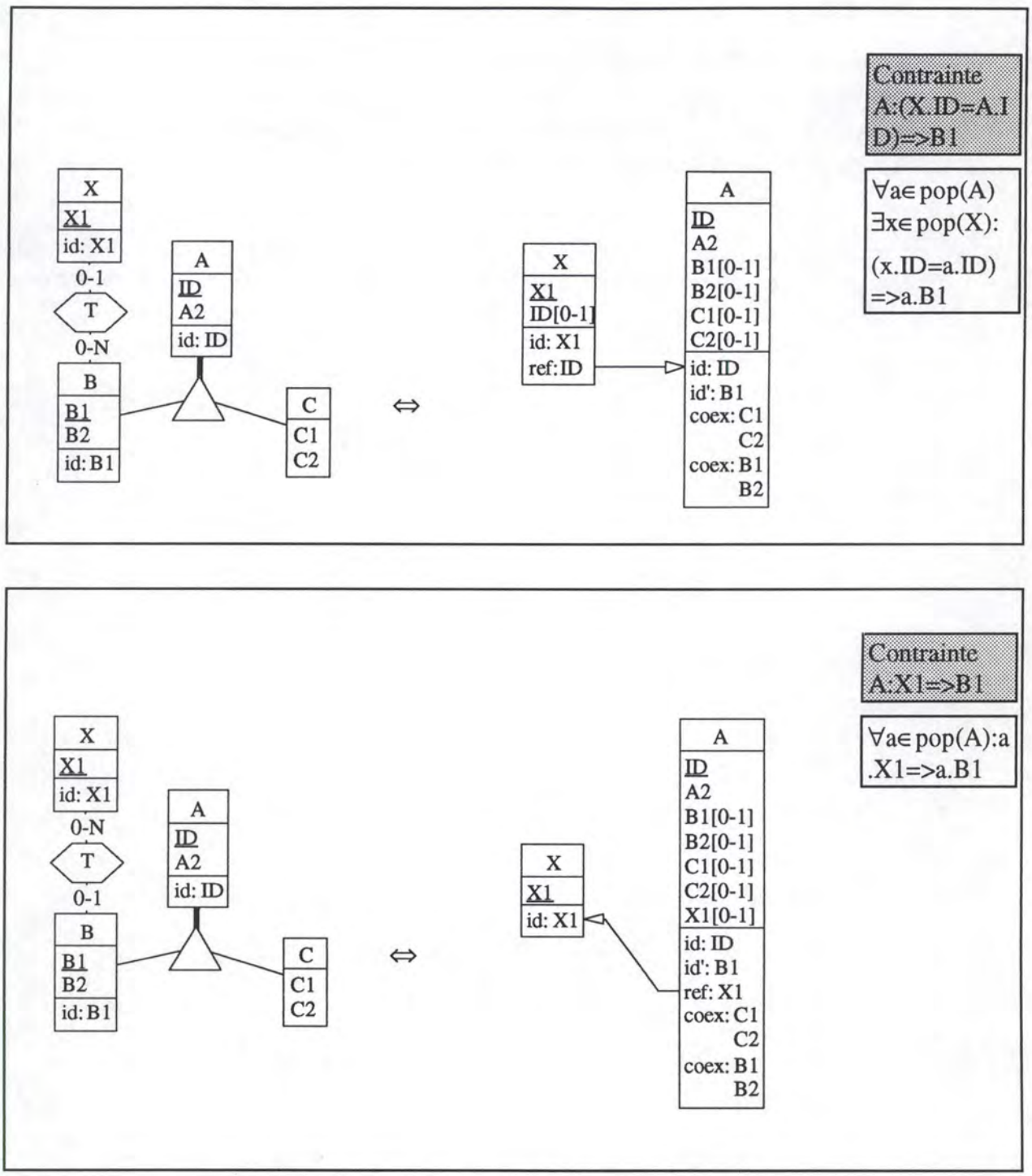


Figure 6 : Transformation par héritage ascendant

La transformation par héritage ascendant n'est pas prévue telle quelle dans l'atelier. Néanmoins, nous pouvons arriver au même résultat par transformation par matérialisation puis par transformation des T.E. sous-types en attributs du surtype.

2.1.1 Code SQL

Si l'atelier permettait cette transformation, le code généré par le générateur SQL serait correct pour la création des tables A et X (Figure 6). Pour permettre la génération des contraintes, le générateur SQL doit être modifié et il doit générer :

Pour le premier schéma,

```
alter table A add constraint IMPL_A
check(not exists
        (select * from X
        where X.IDA = A.ID)
        or (B1 is not null)) ;
```

Pour le deuxième schéma :

```
alter table A add constraint IMPL_A
check((X1 is null)
    or (B1 is not null))
```

2.1.2 Programmation

Dans le cadre de ce mémoire, seule la transformation du deuxième schéma a été prise en compte. Pour cela, la contrainte du deuxième schéma est placée dans la description SEM du schéma et sa syntaxe est du type :

#ISA_IMPL='NOM_GROUPE'{'SURTYPE','NOM_ATTRIBUT_SURTYPE',

'NOM_ATTRIBUT_TE_LIEE'}

Dans l'exemple de la Figure 6, SEM du schéma contient :

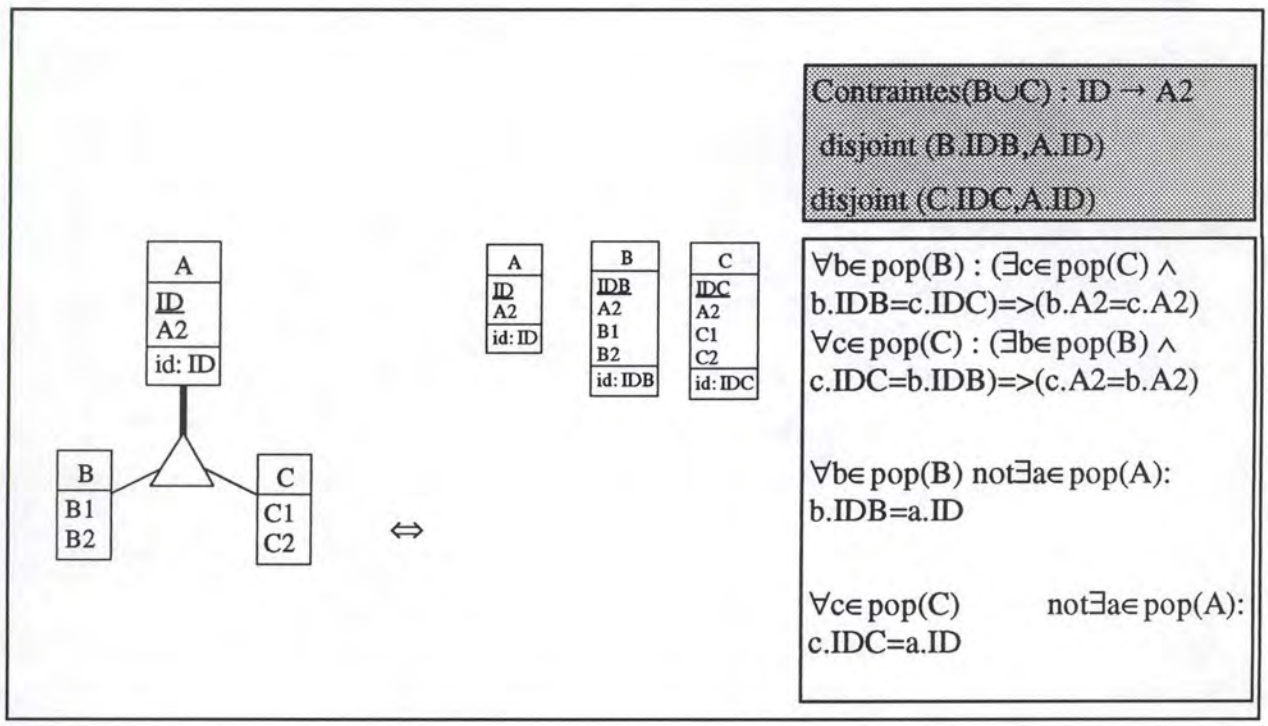
#ISA_IMPL='nom_contrainte'{'A','B1','X1'}

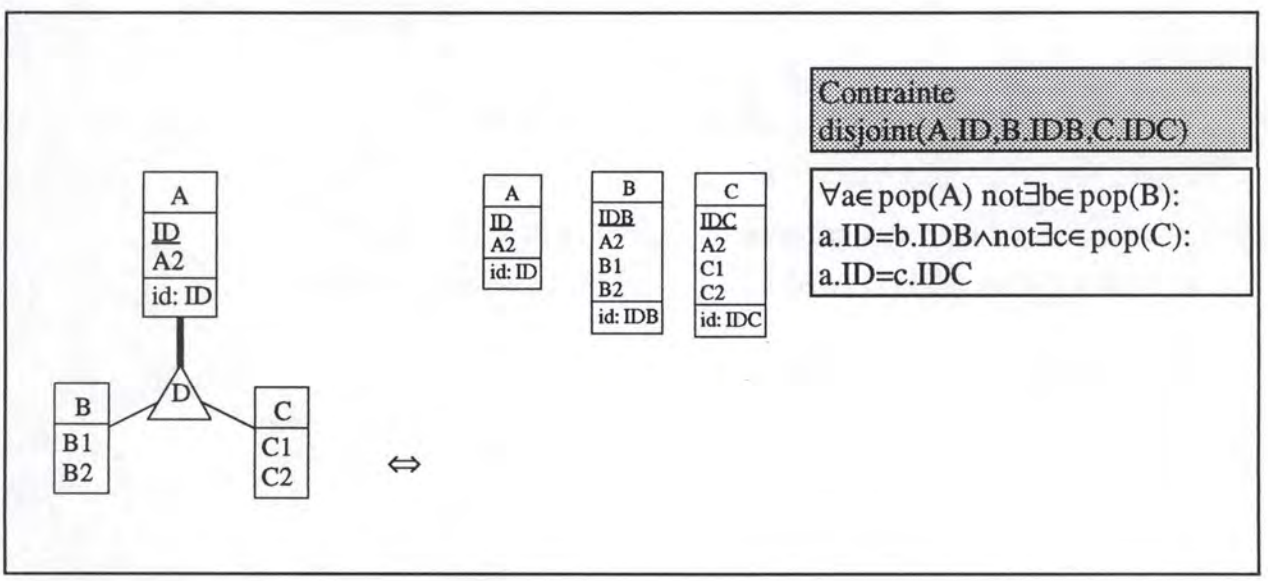
Avec ce string, on peut programmer la génération de la contrainte.

Le code Voyager pour la deuxième contrainte se trouve dans l'annexe C

3. *Transformation par héritage descendant*

Dans le cas de transformation par héritage descendant, certaines relations is-a induisent des contraintes. C'est le cas de relations is-a dont le surtype possède un identifiant.





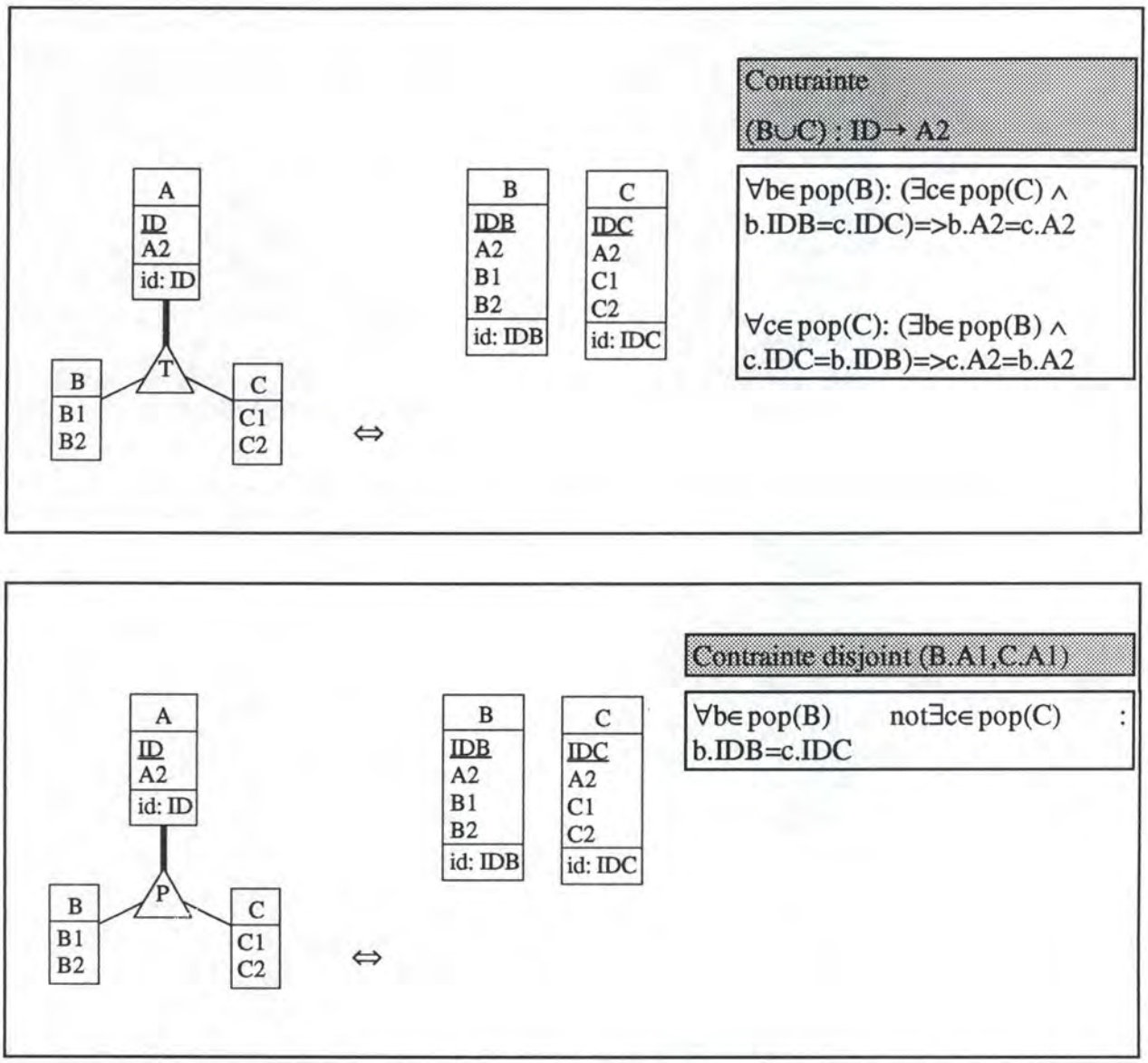


Figure 7 : Transformation par héritage descendant de relations is-a dont le surtype possède un identifiant.

3.1.1 Code SQL

La transformation par héritage descendant n'est pas prévue par l'atelier. Si elle l'était, le code SQL des tables serait correct. Il faudrait encore permettre la génération des contraintes.

Pour la contrainte (B∪C∪D) : IDx→A2,A3 il faut rajouter le code suivant :

```
alter table ? add constraint IMP_B
check(not exists
        (select * from C
        where C.IDC = B.IDB and (C.A2 <> B.A2 or C.A3<>B.A3))
    and (not exists
        (select * from B
        where D.IDD=B.IDB and(D.A2<>B.A2 or D.A3<>B.A3))));
```

```
alter table C add constraint IMP_C
check(not exists
        (select * from B
        where B.IDB = C.IDC and (B.A2 <> C.A2 or B.A3<>C.A3))
    and (not exists
         (select * from D
         where D.IDD=C.IDB and (D.A2<>C.A2 or D.A3 <>C.A3))));

alter table D add constraint IMP_D
check (not exists
        (select * from C
        where C.IDC=D.IDD and (C.A2<>D.A2 or C.A3 <>D.A3))
    and (not exists
         (select * from B
         where B.IDB=D.IDD and (B.A2<>D.A2 or B.A3<>D.A3))));
```

Pour la contrainte disjoint(A.ID,B.IDB,C.IDC), il faut rajouter le code suivant :

```
alter table A add constraint DIS_A_B_C
check(ID not in
      (select IDB from B)
 and ID not in
      (select IDC from C));

alter table B add constraint DIS_B_C_A
check(IDB not in
      (select ID from A)
 and (IDB not in
      (select IDC from C));

alter table C add constraint DIS_C_A_B
check(IDC not in
      (select ID from A)
 and (IDC not in
      (select IDB from B));
```

La contrainte (B∪C∪D) : IDx→A2,A3 doit être exprimée en SQL au moyen des attributs IDx, A2 et A3, provenant du surtype. Pour pouvoir retrouver ces attributs, nous proposons de mettre dans le SEM des attributs de chaque sous-type concernés par ce check, un string de syntaxe :

#ISA_ORIGIN={'TEOR.ATTTEOR'}

où TEOR est le nom du sous-type impliqué dans la contrainte et ATTTEOR est l'attribut de ce TEOR qui correpond.

Par exemple, dans le schéma possédant une contrainte de totalité dans la Figure 7, il y aura dans SEM de l'attribut IDB du T.E. B, une contrainte du type : #ISA_ORIGIN={'C.IDC','D.IDD'} et dans l'attribut A2 du même T.E. : #ISA_ORIGIN={'C.A2','D.A2'}

Dans le schéma ne possédant pas de contrainte, il y aura dans SEM de l'attribut IDB de B : #ISA_ORIGIN={'A.ID'} et dans SEM de l'attribut A2 : #ISA_ORIGIN={'A.A2'}.

Pour retrouver les T.E. concernés par cette contrainte, il est aussi nécessaire d'utiliser la fonction qui retourne les sous-types d'un surtype.

Pour la contrainte disjoint(A.A1,B.A1), la fonction est identique à celle utilisée plus haut pour générer la contrainte de disjonction entre les sous-types dans la matérialisation.

Le code Voyager des contraintes devant être compris dans le code du générateur SQL se trouve dans l'annexe D

V.CONCLUSION

Rappel de l'objectif

Parmi les structures non conformes au modèle relationnel, nous trouvons les relations is-a ou relation de généralisation. Or, peu d'articles et peu d'outils Case proposent des règles complètes pour traduire les relations is-a en structures équivalentes.

Pour cette raison, l'objectif de ce mémoire est de rendre un schéma comportant des relations is-a conforme au modèle relationnel et de générer le code SQL complet.

Etude

Dans ce travail, nous avons réalisé une étude complète des relations is-a avec leur transformation selon les trois techniques de base, à savoir la matérialisation, l'héritage ascendant et l'héritage descendant.

A travers cette étude, nous avons pu remarquer que la simplicité des relations is-a contraste beaucoup avec la complexité de leur traduction complète.

Problème

La transformation des relations is-a amène très souvent des contraintes. Il est nécessaire de les exprimer dans le schéma et ensuite de permettre leur traduction en SQL.

Le choix effectué pour l'expression de ces contraintes pose un problème : les contraintes ne sont pas directement visibles sur le schéma.

Moyen

Le raisonnement élaboré pour cette étude est de généraliser les cas de relations is-a rencontrés. Ainsi, grâce à l'étude systématique des relations is-a, chaque cas de relation is-a que l'on rencontrera, pourra se rapprocher d'un des cas étudiés.

Pour cela, nous avons pris cinq exemples de relations is-a dont nous avons pu tirer les éléments nécessaires pour pouvoir effectuer des transformations. Ces transformations amènent à des schémas conformes au modèle relationnel et les contraintes rencontrées dans un schéma sont ensuite exprimées en SQL. Ce code SQL que nous avons choisi de générer n'est valable que pour des systèmes de gestion de bases de données qui, comme Interbase, acceptent des checks mentionnant d'autres tables.

Si l'on désire un code SQL compatible avec un autre SGBD, il est possible d'utiliser des méthodes plus largement acceptées.

Implémentation

L'outil central utilisé est l'atelier DB-Main. Il permet les transformations de schémas et la génération des contraintes additionnelles ainsi que la génération du code SQL qui y correspond. La programmation de ces générations de code SQL est réalisée en *Voyager 2*, langage utilisé dans l'atelier DB-Main.

Résultats

Les transformations de relations is-a que l'on a étudiées ont pour but d'être implémentées dans l'atelier DB-Main. Jusqu'à présent, seules les transformations par matérialisation et par héritage ascendant sont implémentées. Il pourrait être intéressant d'automatiser les autres transformations des relations is-a et de généraliser l'expression des contraintes dans le schéma. Le code Voyager concernant la génération du code SQL est prévu pour être modifié en fonction de l'orientation choisie pour l'implémentation des transformations dans l'atelier.

Remarquons encore que seul l'héritage simple a été pris en compte. Une extension possible à ce mémoire est d'étudier l'héritage multiple des relations is-a, où un sous-type peut avoir plusieurs surtypes.

VI.Bibliographie

[1] C Batini ; S Ceri ; S B Navathe, *Conceptual database design - An entity-relationship approach,* Benjamin/Cummings, 1992.

[2] F. Bodart ; Y Pigneur, *Conception assistée des systèmes d'information méthode modèle - outils,* Masson, Paris ; Milan ; Amsterdam, 1994.

[3] C J Date, *An introduction to database systems volume 1,* Addison-wesley, 1995.

[4] C J Date, *An introduction to database systems volume 2,* Addison-wesley, 1985.

[5] V Englebert, *Voyager 2 reference manual version 2 release 1,* FUNDP, Namur, Belgique, 1996.

[6] V Englebert, *Voyager 2 (version 3.0) - reference manual,* FUNDP, Namur, Belgique, 1997

[7] J-L Hainaut, *Introduction à SQL, un système de gestion de bases de données relationnelles,* FUNDP, Namur, Belgique, 1992.

[8] J-L Hainaut, *Conception d'une base de données - démarche de conception,* FUNDP, Namur, Belgique, 1995.

[9] J-L Hainaut, *Transformation-Based database engineering,* FUNDP, Namur, Belgique, 1995.

[10] J-L Hainaut ; J-M Hick ; V Englebert ; J Henrard ; D Roland, "Understanding the implementation of IS-A relations", *Research Report,* FUNDP, Namur, Belgique, 1996

[11] J-L Hainaut, J-M Hick, V Englebert, J Henrard, D; Roland, *The DB-Main database Engineering Case tool (Version 2) Functions overview,* FUNDP, Namur, Belgique, 1997.

[12] J-L Hainaut, J-M Hick, V Englebert, J Henrard, D; Roland,, *DB-Main Tutorial volume 1 Introduction to Database Design,* FUNDP, Namur, Belgique,1997.

[13] J-L Hainaut, *Analyse et conception de Bases de données - Matière approfondie,* FUNDP, Namur, Belgique, 1997.

[14] J-L Hainaut, J-M Hick, V Englebert, J Henrard, D; Roland,, *The DB-main database Engineering Case tool (version 3) functions overview,* FUNDP, Namur, 1997.

ANNEXE A

```
/*************************************************************************/
/*     Fonction qui renvoit la liste des sous-types d'un type d'entites  */
/*************************************************************************/
/* Cette fonction ne renvoit la liste des sous-types que lorsque les
  contraintes referentielles vont des sous-types vers le super-type. Le cas
  inverse n'est pas a prendre en compte. */
function list ConstruireListeSousTypes(entity_type: sup)
string:          val;
data_object:     dto1, dto2;
entity_type:     orent;
group:           targr, orgr;
constraint:      cst;
member_cst:      mcst1, mcst2;
list:            lsub;
{
 lsub := [];
 dto1 := sup;
 for targr in GROUP[targr]{@DATA_GR:[dto1]} do {
  for mcst1 in MEMBER_CST[mcst1]{@GR_MEM:[targr]} do {
   if mcst1.mem_role=TAR_MEM_CST then {
    for orgr in GROUP[orgr]{GR_MEM:MEMBER_CST[mcst2]
     {@CONST_MEM:CONSTRAINT[cst]{CONST_MEM:[mcst1]}} with targr<>orgr}
do {
      val := GetProperty(orgr.sem,"subtype");
      if (val <> ("\n#not-found")) and (val <> ("\n#corrupted")) then {
       orent:=GetFirst(DATA_OBJECT[dto2]{DATA_GR:[orgr]});
       AddLast(lsub,orent);
      }
     }
   }
  }
 }
 return lsub;
}

/*************************************************************************/
/* Fonction qui imprime un super-type avec sa contrainte de sous-typage et */
/* ses sous-types
*/
/*************************************************************************/
procedure ImprimerSousTypage(entity_type: sup, string: typ, list: lsub)
entity_type: ent;
cursor:      c;
{
 print(["Super-type: ",sup.name,"\n","Constraint type: \"",typ,"\"\n",
       "Sub-types: "]);
```

```
  attach c to lsub;
  while IsNoVoid(c) do {
   ent := get(c);
   print(ent.name);
   c>>;
   if IsNoVoid(c) then { print(", "); }
  }
  print("\n\n");
}
/*************************************************************************/
/* Procedure qui recherche l'identifiant d'un T.E donne
*/
/*************************************************************************/
function string RechercherId(entity_type:ent)
group:          gr;
cursor:         c;
integer:        typ;
attribute:      at;
list:           l;

{for gr in GROUP[gr]{@DATA_GR:[ent]} do
{
  if gr.primary then
  {
     l:=GetListOfComponents(gr);
     attach c to l;
     while IsNoVoid(c) do
       {
        typ:=GetType(get(c));
        if(typ=SI_ATTRIBUTE) then
           {
            at:=get(c);
            return at.name;
           }
       }
  }
}
}
/*************************************************************************/
/* Procedure qui imprime le check de la contrainte de disjonction        */
/*************************************************************************/
procedure Imprime CheckDis(entity_type:sup,list:lsub)
entity_type :    ent,ent2;
cursor:         c,cc;
string :        ident,ident2;
integer:        bool;
{
  attach c to lsub;
  while IsNoVoid(c) do {
```

```
    ent := get(c);
    print(["alter table ",ent.name," add constraint "," nom_contrainte \n",
            "check( "]);
    attach cc to lsub;
    while IsNoVoid(cc) do {
     ent2:=get(cc);
     if (ent2<>ent) then {

       ident:=RechercherId(ent);
          ident2:=RechercherId(ent2);
       print([ident," not in(select ",ident2," from ",ent2.name,")" ]);
       }
     cc>>;
     bool:=0;
     if (ent2=ent) and (GetFirst(lsub)=ent) then
         {bool:=1;}
     if IsNoVoid(cc) then {
      if (get(cc)<>ent and bool=0) then {
        print("\n  and ");
        }
     };
     if IsVoid(cc) then {
          print (");\n");
     }


    }
    print("\n");
    c>>;

  }
}

/*************************************************************************/
/* Procedure qui imprime le check de la contrainte de totalite
*/
/*************************************************************************/
procedure ImprimerCheckTot(entity_type:sup,list:lsub)
entity_type :   ent;
cursor:         c;
string :        ident,ident2;
{
 attach c to lsub;
 print(["alter table ",sup.name," add constraint "," nom_contrainte \n",
           "check( "]);
 ident:=RechercherId(sup);
 while IsNoVoid(c) do
   {
   ent := get(c);
```

```
    ident2:=RechercherId(ent);
    print([ident," in(select ",ident2," from ",ent.name,")" ]);
    c>>;
    if IsNoVoid(c) then
      {
       print("\n   or ");
       }
    if IsVoid(c) then
      {
      print (");\n");
       }
    }
}


/***********************/
/* Programme principal      */
/***********************/

begin
  sch:=GetCurrentSchema();
  if not(IsVoid(sch)) then {
    SetPrintList("","","");
    for dto in DATA_OBJECT[dto]{@SCH_DATA:[sch] with
GetType(dto)=ENTITY_TYPE} do {
     l := [];
     ent :=dto;
     val := GetProperty(ent.sem,"supertype");
     if (val <> ("\n#not-found")) and (val <> ("\n#corrupted")) then {
      l := ConstruireListeSousTypes(ent);
      if (Length(l)) then {
        ImprimerSousTypage(ent,val,l);
           if val="D" then{
           ImprimerCheckDis(ent,l);}
           if val="T" then{
              ImprimerCheckTot(ent,l);}
        if val="P" then{
              ImprimerCheckDis(ent,l);
           ImprimerCheckTot(ent,l);}
          }
     }
    }
  }
```

ANNEXE B

```
/*************************************************************************/
/* Procedure qui imprime le check de la contrainte pour l'attribut TYPE  */
/*************************************************************************/
procedure ImprimerCheckType(entity_type:sup,list:lsub)
entity_type :   ent;
cursor:         c;
string :        ident,ident2;
attribute :     att;
string:         atype;
{
 attach c to lsub;
 ident:=RechercherId(sup);
 while IsNoVoid(c) do
  {
   ent := get(c);
   print(["alter table ",ent.name," add constraint "," nom_contrainte \n",
          "check(not exists "]);
   ident2:=RechercherId(ent);
   for att in ATTRIBUTE[att]{@OWNER_ATT:[sup] with
        StrFindSubStr(att.sem,0,"#ISA_TYPE=")>=0} do
    {atype:=att.name;}
   print(["\nselect",atype, "from ",sup.name," where ",sup.name,".",
       ident," = ",ent.name,".",ident2," and TYPE not like '%",
       ent.name,"%'));\n\n" ]);
    c>>;
 }
}
```

ANNEXE C

```
schema:sc;
string:sema;
string : surtype,attsurtype,attteliee;
integer:lentete,debsurtype,debattsurtype,debatteliee;
string:nomcontrainte;
begin
  lentete:=10;
  /*On prend le schéma courant et le sem du schéma*/
  sc:=GetCurrentSchema();
  sema:=sc.sem;
  lentete:=9;
  /*On va découper SEM dans les bons champs*/
  /*On met le nom_contrainte dans nomcontrainte*/
  nomcontrainte := StrGetSubStr(sema,lentete+2,StrFindChar(sema,0,'{')-lentete-3);
  debsurtype:=StrLength(nomcontrainte)+lentete+5;
  surtype := StrGetSubStr(sema,debsurtype,StrFindChar(sema,debsurtype,',')-debsurtype-1);
  debattsurtype:=debsurtype+StrLength(surtype)+3;
  attsurtype := StrGetSubStr(sema,debattsurtype,StrFindChar(sema,debattsurtype,',')-
        debattsurtype-1);
  debatteliee:=debattsurtype + StrLength(attsurtype)+3;
  attteliee := StrGetSubStr(sema,debatteliee,StrFindChar(sema,debatteliee,'}')-debatteliee-
        1);
  /*On génère le check*/
  print(["alter table ",surtype," add constraint ",nomcontrainte,
        "\ncheck((",attteliee," is null)\nor(",attsurtype," is not null));"]);
end
```

ANNEXE D

```
/*************************************************************************/
/* Procedure qui imprime le check de la contrainte INCL                  */
/*************************************************************************/
procedure ImprimerCheckIncl(entity_type:sup,list:lsub)
entity_type :   ent2;
cursor:         cc;
schema:         sc;
string :        table,table2,nomtable,nomtable2, nomattr,nomattr2;
attribute:      ident,att;
integer:        bool,entete,r,i;
string:         sema;
data_object:    data;

{
 sc:=GetCurrentSchema();
 entete := 14;
 attach cc to lsub;
 while IsNoVoid(cc) do
{
 ent2 := get(cc);
 ident:=RechercherAttId(ent2);
 if ident.name<>"" then
  {
  sema:=ident.sem;
  debut:=0;
  r:=StrFindSubStr(sema,debut,"ISA_ORIGIN={");
  while r>=0 do
    {
     table:=StrGetSubStr(sema,debut+entete,StrFindChar(sema,debut,'}'));
     nomtable:=StrGetSubStr(table, 0,StrFindChar(table,0,'.'));
     nomattr:=StrGetSubStr(table,StrFindChar(table,0,'.')+1,StrFindChar(table,0,'")-
                StrFindChar(table,0,'.')-1);
     for data in DATA_OBJECT[data]{ @SCH_DATA:[sc] with
                GetType(data)=ENTITY_TYPE} do
       {if data.name=nomtable then
         {
           if RechercherId(data)=nomattr then
             {
              print(["alter table ",ent2.name," add constraint "," nom_contrainte \n",
                   "check(not exists \n(select * from "]);
             bool:=0;
              print([nomtable," where ",nomtable,".",nomattr," = ",ent2.name,".",ident.name,"
             and ","("]);
```

```
            for att in ATTRIBUTE[att]{ @OWNER_ATT:[ent2] with
                  (StrFindSubStr(att.sem,0,"#ISA_ORIGIN=")>=0 and
                  att.name<>ident.name)} do
             {
               if bool>=1 then
                 {print(" or ");     }
               table2:=StrGetSubStr(att.sem,entete,StrFindChar(att.sem,0,'}'));
               nomtable2:=StrGetSubStr(table2, 0,StrFindChar(table2,0,'.'));
               nomattr2:=StrGetSubStr(table2,StrFindChar(table2,0,'.')
                          +1,StrFindChar(table2,0,"')-StrFindChar(table2,0,'.')-1);
               print([nomtable2,".",nomattr2,"<>"]);
               print([ent2.name,".",att.name,"  "]);
               bool:=bool+1;
               }
             print("))");
             }
           }
         }

       print(");\n");
       debut:=debut + StrLength(nomtable) + StrLength(nomattr)+1+4+ entete;
        r:=StrFindSubStr(sema,debut,"ISA_ORIGIN={");
      }
    }
   cc>>;
  }
}
```